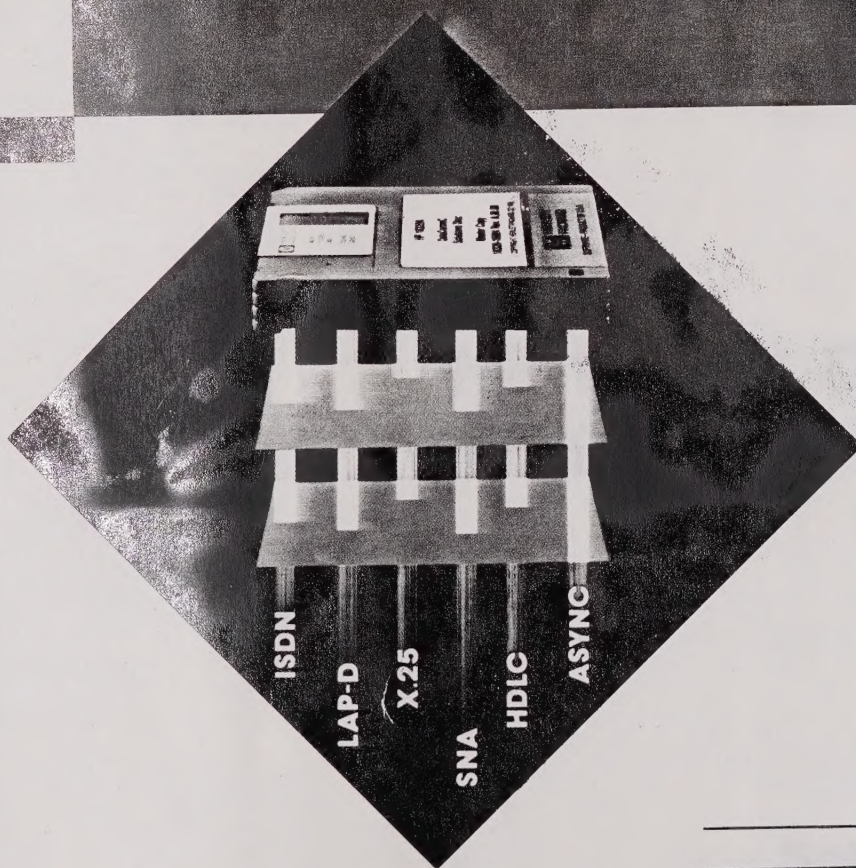




HEWLETT
PACKARD



Solutions
Program



HP 18320A DataCommC
Programming Language
for the HP 4954A Protocol Analyzer

RETURN THIS MANUAL

**IF NOT RETURNED,
YOU WILL BE CHARGED
A MINIMUM OF \$100.00**

PLEASE RETURN TO:
IR INSTRUMENT SERVICES
INTERNATIONAL DISTRIBUTION CENTER
1830 WEST AIRFIELD DRIVE
DFW AIRPORT, TX 75261



Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_05518

HP 4954A Protocol Analyzer

Contents

Notices

HP 18320A DataCommC Programming Language

Solution Programs



HEWLETT
PACKARD

Manual Part Number: 18320-99506
Microfiche Part Number: 18320-98813

Printed in U.S.A. SEPTEMBER 1990
E0990

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

If your software application or hardware should fail, contact your local Hewlett-Packard Sales Office listed in the protocol analyzer operating manual.

© Copyright 1990 Hewlett-Packard Company.
Colorado Telecommunications Division
5070 Centennial Boulevard
Colorado Springs, CO, 80919-2497

Contents

1 DataCommC Solution Programs

Modifying Programs	1-2
Source and Include File Listings	1-2
Displaying or Printing Programs.....	1-3
Installing the Solution Program Files.....	1-4

2 Automatically Running Programs

AR Files.....	2-1
Timed Start Menu.....	2-2
TSM Format.....	2-2
TSM Criteria	2-3
Program Character String	2-3
Start Time Character String.....	2-3
Start Date Character String.....	2-4
Typical Timed Start Menu.....	2-4
Messages	2-5
Running the AR Program.....	2-6
AR.linkcommand File.....	2-6
Creating the Program File.....	2-6
Auto Run Menu Source File	2-8
AR_MENU.csource.....	2-10
s_type() routine	2-10
get_filename() routine.....	2-12
getval() routine.....	2-13
gettime() routine	2-14
getdate() routine	2-15
convert_menu() routine	2-16
read_menu() routine.....	2-20
init_menu() routine.....	2-22

2 Automatically Running Programs (Continued)

Auto Run Timing File	2-22
AR_RUN.csource.....	2-23
same_date() routine.....	2-23
same_time() routine	2-24
run_menu() routine	2-24
Auto Run User Interface File	2-26
AR_UI.csource	2-26
main() routine	2-26
Auto Run Include File.....	2-28
AR_H.include	2-28
AR_TEST.csource File Listing.....	2-30
AR_TEST.csource	2-31

3 Decoding Displays

Display Decoding Files.....	3-2
Creating a Display Decode Program.....	3-2
BOPS DECODE Decoding Structure	3-4
decode_frame().....	3-4
display_frame()	3-4
Modifying Frame Decoding	3-5
Adding Frame Information	3-5
The Formatted Display Routine	3-6
Examples.....	3-8
Modifying the Display.....	3-8
The Softkey Structure	3-9
BLANK Type Softkey.....	3-9
OFF/ON Type Softkey.....	3-10
CYCLE Type Softkey	3-11
ZCYCLE Type Softkey.....	3-12
LAYER Type Softkey.....	3-13
Modifying Softkeys	3-14
Running BOPS DECODE.....	3-15
Defining Linkages.....	3-15
Creating the Program File.....	3-16
Source and Include File Listings	3-16
Program Listing - MAIN_DEC.csource	3-17
main_dec() routine.....	3-17

3 Decoding Displays (Continued)

Program Listing - GBOPS.csource.....	3-20
decode_frame() routine.....	3-21
LAPB_U() routine.....	3-28
LAP_U() routine.....	3-29
HDL_C_U() routine.....	3-30
SDLC_U() routine.....	3-32
Program Listing - GBOPS_DEC.csource.....	3-33
display_frame() routine.....	3-34
Program Listing - GBOPS_KEYS.csource.....	3-41
gbopsL2_keys[] structure.....	3-41
gbopsL3_keys[] structure.....	3-41
gbops_top[] structure.....	3-41
Program Listing - GBOPS_LEAD.csource.....	3-42
lead_change_disp() routine.....	3-42
lead_stat_disp() routine.....	3-47
Program Listing - USER.csource.....	3-56
user() routine.....	3-56
Program Listing _DECODE.include.....	3-57
Program Listing - MACROS.include.....	3-57
Program Listing - GBOP.include.....	3-58
Program Listing - GLOBUS.include.....	3-61

4. Transmitting a Frame

Frame Transmission Files.....	4-1
Running the Program.....	4-2
MAKE.text Files.....	4-2
Linkcommand Files.....	4-4
Messages.....	4-5
Source and Include File Listings.....	4-6
Program Listing - PROCESS1.csource.....	4-6
initialize() routine.....	4-7
configure() routine.....	4-8
spawn_others() routine.....	4-8
initial_and_sync() routine.....	4-9
packetize() routine.....	4-9
sendframe() routine.....	4-10
get_command_resp() routine.....	4-11
get_PF_bit() routine.....	4-12
get_address() routine.....	4-13
get_string() routine.....	4-14

4. Transmitting a Frame (Continued)

Program Listing - PROCESS1.csource (Continued)

update_rxcount() routine.....	4-15
unnumbered() routine	4-16
information() routine	4-18
supervisory() routine.....	4-19
menu_zero() routine	4-20
menu_one() routine	4-20
menu_two() routine.....	4-22
main() routine.....	4-23
Program Listing - PROCESS2.csource	4-24
initial_and_sync() routine.....	4-25
extractframe() routine.....	4-26
get_frame_type() routine.....	4-27
print_address() routine	4-28
print_control() routine.....	4-28
print_PF() routine	4-30
expand_control() routine.....	4-30
print_extras() routine.....	4-32
print_type() routine.....	4-32
print_fcs() routine.....	4-33
disframe() routine	4-34
main() routine	4-35
Program Listing - PROCESS3.csource	4-37
initialize() routine.....	4-37
extractpacket() routine	4-38
disppacket() routine	4-39
main() routine.....	4-40
Program Listing - PROCESS23.csource	4-41
setscreen() routine.....	4-41
Program Listing - PROCESS_H.include	4-42

5. Collecting Statistics

Running the Program	5-1
Statistics Program Display	5-2
Selecting the Data Source	5-3
On Line Data Source	5-3
Buffer Data Source.....	5-3
Source File Listing	5-4

5. Collecting Statistics (Continued)

Program Listing - CUSTOM_COU.csource	5-4
main() routine.....	5-6
decode() routine	5-8

Figures

2-1. AR_EXAMPLE.text File	2-5
2-2. AR.linkcommand File.....	2-6
2-3. MAKE_AR.text File	2-7
3-1. Sample Decoding Program	3-3
3-2. BOPS.linkcommand File	3-15
3-3. GBOPS.text File	3-16
4-1. MAKE1.text File Listing	4-3
4-2. MAKE2.text File Listing	4-3
4-3. MAKE3.text File Listing	4-3
4-4. PROCESS2.linkcommand File.....	4-4
4-5. PROCESS3.linkcommand File.....	4-4
5-1. FCS Statistics Display	5-2

Tables

1-1. Installing Solution Program Files.....	1-4
3-1. Rules for Using the dispf() Routine	3-6
3-2. dispf() Routine Description	3-7

Printing History

New editions are complete revisions of the manual. Update packages (formerly known as "Manual Changes") are issued between editions. They contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change when an update is incorporated.

Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correlation between product updates and manual updates.

Edition 1.....	February 1989
Edition 2.....	September 1990

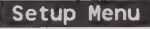
Problems? Call for Help

If you have problems operating your HP protocol analyzer or any of the applications, call the CTD Customer HelpLine or use your fax machine to send your questions.

HelpLine - (719) 531-4567
Fax - (719) 531-4505

Syntax Conventions

The following symbols, abbreviations, and other conventions are used in this publication.

 	A control character entry from the keyboard where both the  key and an alphanumeric key are pressed at the sametime. To enter CONTROL U press  and  .		
 softkey	A keyboard entry where both the  and a softkey are pressed at the same time to select an auxiliary softkey function.		
FILENAME	Within menus or screens, a parameter that must be entered in the exact format shown.		
filename	Within menus or screens, a user-defined parameter.		
	A softkey.	10_2 or 10 binary	A base 2 number.
	A keyboard command entry.	10_8 or 10 octal	A base 8 number.
[SPACEBAR]	The keyboard spacebar.	10_{10} or 10 decimal	A base 10 number.
   	Cursor control keys.	10_{16} or 10 hex	A base 16 number.

Warning	An operating procedure, practice, etc., which, if not correctly followed, could result in personal injury or loss of life.
----------------	--

Caution	An operating procedure, practice, etc. which, if not strictly observed, could result in damage to, or destruction of, equipment or software.
----------------	--

Note	Explanatory comments or supplementary instructions.
-------------	---

DataCommC Solution Programs

The DataCommC Solution Programs are a set of C files for basic network testing. They are not limited to only one application as are most "canned tests." Instead these programs are a platform, not a ceiling, on what you can test. You can easily modify each of them to meet your needs for network management or analysis. Subsequent chapters discuss each program in detail to include the uses, characteristics, capabilities, and file listings. The DataCommC solution programs furnished are listed below.

- Automatically Running Programs (see chapter 2).
- Decoding Displays (see chapter 3).
- Transmitting a Frame (see chapter 4).
- Collecting Statistics (see chapter 5).

The source and include files are copyrighted and their usage is subject to the following restrictions.

NOTICE

All rights reserved. Reproduction, adaptation, or translation, without prior written permission is prohibited, except as allowed under the copyright laws.

Modifying Programs

Specific modifications to individual programs are discussed in subsequent chapters. In general, the following rules apply to all modifications.

1. *DO NOT* modify programs on the Solutions disc. This is the master disc and should only be used for installing the software.
2. You can modify programs *after* they are loaded on the hard disc.
3. Use the hints described in "Displaying or Printing Programs," below.
4. Solution programs are *WORM* programs. They may be **Written Once**, but they are **Read Many** times. Comments will aid others in understanding what a program or a routine does.

Source and Include File Listings

A listing of each Solutions disc source and include file is included. These listings have been reformatted for easier reading and the comments expanded to better describe program functions and operations. *The program code in these listings and the program code on the disc are identical at the time of publication.* The comments and organization are not identical.

As with all software products, the programs may be revised prior to shipment. Therefore, it is possible that there may be *minor* differences.

Displaying or Printing Programs

You can display or print these programs using the DataCommC editor. Both the analyzer screen and the ThinkJet and QuietJet printers normally display about 80 characters on a line.

With the analyzer cursor control key (⏏) you can scroll past column 80 to the end of the line. The printers are a different matter. The ThinkJet will truncate all lines containing more than 80 characters. On the QuietJet printer you can select the smallest type font and display approximately 160 characters per line.

HINT

Keep the length of *program lines* to a reasonable, easily viewable length. Approximately 80 characters is recommended. Lines with comments can be longer providing you keep in mind that lines longer than 100 characters are difficult to read on the screen and almost impossible to print.

HINT

When writing or editing programs change the *default* tab setting to 3 *before* entering any data or modifications. (Press **Escape**, **Define**, **Fixed**, **3**, and **Return**.) This saves five spaces for per indentation level.

HINT

When entering continuation lines, use the *same number* of tabs as on the first line and then use spaces to line up the text.

Installing the Solution Program Files

The procedure for installing the solution program files (see table 1-1) is the same as for installing DataCommC.

Table 1-1. Installing Solution Program Files

1. Apply power to the HP 4954A and insert the *Solutions Disc* in the flexible disc drive.
2. Select the *auto_load* directory and use **Disable Auto Load** on all files.
3. Press **Shift** **Reset** to reset the analyzer.
4. Press, in sequence, **Mass Store**, **Change Directory**, **Select Unit**, **Flexible Disc**, and **Execute**.

The analyzer displays the flexible disc directory.

5. Press, in sequence, **Exit Operation**, **Load File**, and **Select File**.
6. Use **Next File** and **Previous File** to highlight the *Install* application module.
7. Press **Execute** and follow the instructions on the display.
8. After all files are installed the analyzer displays the top level menu. Press, in sequence, **Mass Store**, **Change Directory**, **Select Unit**, **Hard Disc**, **Execute**, and **Exit Operation** to return to the hard disc top directory.

Automatically Running Programs

The function of the Auto Run (AR) solution program is to automatically start DataCommC programs at specific times on selected dates. The user-created Timed Start Menu (TSM) defines the program, the time, and the date to run a program. Each TSM controls as many as five programs. The conditions for running a program are as follows.

- Run a program a *maximum* of five times.
- Run a program at the same time every day.
- Run a program on five different dates with a different start time each date.
- Run a program using a combination of start times and dates.

AR Files

The AR program uses the following files that are on Solutions Disc 1.

AR_MENU.csource	Selects the TSM and translates it into the data base in <i>current_menu</i> .
AR_RUN.csource	Checks the real time clock and initiates a program at the time and on the date specified in the TSM.
AR_UI.csource	Loads and runs the TSM.
AR_H.include	Contains definitions for automatically running programs.
AR_EXAMPLE.text	A sample file for developing TSMs.
AR_MAKE.text	This ASCII file specifies the object files, program name, and the sequence for running the program.
AR.linkcommand	This ASCII file defines the locations of the files used by the MAKEFILE.text file.
AR_TEST.csource	A file for testing TSMs.
AR_README.text	This file is a synopsis of the material contained in this chapter.

timed start menu.text

This user-created ASCII file defines the following.

- The programs to be run.
- The dates to run *each* program.
- The time to start *each* program on *each* date.

Timed Start Menu

You can tailor the TSM to meet your requirements for network operations and analysis. Should you need to run more than five programs you can either modify the *AR_H.include* file to process more programs or you can create additional TSMs. However, the *AR.program* can only use only one TSM at a time.

TSM Format

The basic format of a TSM file (see figure 2-1) is as follows.

- All entries start in column 1.
- TSM is not case sensitive. It interprets uppercase and lower case letters as the same character.
- The entry on the first line *must be* **<tmf>;**
- TM key characters (i.e., the character in column 1) are as follows:
 - d** Start of the date character string.
 - de** Start of the *every* day character string.
 - f** Start of the program character string.
 - t** Start of the time character string.

Any string of characters beginning with a *non-reserved TM character*, beginning in any column besides column 1, or after the semi-colon terminator on a character string line, is interpreted as a comment string and is not processed by the program.

- *All* character strings are terminated with a semi-colon (;).
- Each program routine is composed of a *minimum* of three character strings - program, time, and date.
- For each start time there must be a corresponding start date.

TSM Criteria

When creating the TSM be sure that the programs meet the following criteria.

- The program name, to include the device, path name, and file name, are correct.
- The specified program does exist.
- Programs that are scheduled to run at the same time are compatible.

In the first two cases, when AR cannot find the specified program, it assumes the program does not exist and displays an error message at the time you scheduled the program to start.

An incompatibility between programs includes starting a program that does not share the CPU and subsequently starting another program *before* this program is finished. The second program does not run.

Program Character String

The first entry in each program routine is the *program character string* in the format:

f *device:path/file name*;

where:

- | | | |
|------------------|---|--|
| <i>device</i> | = | Disk drive designator (e.g., a, c, d, etc.). |
| <i>path</i> | = | Path name to include all directories and sub-directories (e.g., C/User/User1/) |
| <i>file name</i> | = | Case sensitive program name to include the extension (e.g., test1.program). |

A typical program character string is: **f** c:C/User/User1/test1.program;

Start Time Character String

The second entry in each program routine is the *start time* character string. There may be as many as five different start time character strings, each on a separate line.

The start times are read sequentially starting with the first character string. When this time is the same as the protocol analyzer clock time, the *corresponding* date character string is checked. If the date matches the analyzer date, the program can start.

The format for the time character string is as follows.

t *hh:mm*;

where:

- hh* = The hour to start the program using a 24-hour clock
(e.g., 2 pm is 14, 8 pm is 20, etc.).
- mm* = The minute to start the program.

A typical time character string is: **t 16:48**;

Start Date Character String

The third type of character string entered for each program is the *start date* character string. This string applies the *last* program string defined and the time character string that occurs in the same order (e.g., the first start date string is used with the first start time character string; the third date string pertains to the third time string, etc.).

There are two formats for the start date character string. For a program to start on a specific date the format is as follows.

d *mm/dd/yy*;

where:

- mm* = The month when the program is to start where 1 = January and
12 = December.
- dd* = The date on which the program is to start (i.e., 1 through 31).
- yy* = The last two digits of the year when the program is to start
(e.g., 89 = 1989; 90 = 1990, etc.).

A start date character string to start a program on January 24th, 1989 is: **d 1/24/89**;

For a program to run every day at the same time, the start date character string is: **de**;
There are no other parameters for the start every day character string.

Note

There must be one-to-one correlation between start time character strings and start date character strings.

Typical Timed Start Menu

The following is an example of a typical timed start menu. Note that this ASCII file does not require any annotations before or after a comment (e.g., Execute the specified . . . 24th, First program, Second program, etc.), regardless of where they appear in the file.

```

<tmf>;
Execute the specified program every day at 8:00 am and at 2:56 pm on January 24th.
f c:C/User/PGM/test1.program;           First program
t 14:56;                                Run at 2:56 PM on Jan 24, 1989
d 1/24/89;
t 8:00;                                  Run everyday at 8:00 AM
de;
Execute this program on five successive days at different times.
f c:C/User/PGM/test2.program;           Second program
t 1:56;                                Run at 1:56 AM on Jan 24, 1989
t 2:56;                                Run at 2:56 AM on Jan 25, 1989
t 3:56;                                Run at 3:56 AM on Jan 26, 1989
t 4:56;                                Run at 4:56 AM on Jan 27, 1989
t 5:56;                                Run at 5:56 AM on Jan 28, 1989
d 1/24/89;
d 1/25/89;
d 1/26/89;
d 1/27/89;
d 1/28/89;
Execute this program four times every day and once on February 3.
f c:C/User/PGM/test3.program;           Third program
t 8:00;                                Run everyday at 8:00 AM
t 9:00;                                Run everyday at 9:00 AM
t 10:00;                               Run everyday at 10:00 AM
t 11:00;                               Run everyday at 11:00 AM
t 12:00;                               Run at 12:00 PM on Feb 3, 1989
de;
de;
de;
de;
d 2/3/89;

```

Figure 2-1. AR_EXAMPLE.text File

Messages

The *AR.program* generates the following messages that describe the status of the program.

- Error: A date was incorrectly specified for Program file: *file name*
- Error: A mismatched number of start times and dates were specified for the Program file: *file name*.
- Error: A program file name was incorrectly specified: *file name*
- Error: A start date was specified before any Program files.
- Error: A start time was specified before any Program files.
- Error: Either no start date or time was specified for the Program file: *file name*
- Error: More the 5 Program files were specified.
- Error: No program files were specified.

Error: Too many start dates were specified for Program file: *file name*
Error: Too many start times were specified for Program file name: *file name*
Errors were encountered converting the Timed Start Menu. Press Exit to Continue.
The Menu file could not be opened - Try again or Exit
The Menu file did not start with <tmf>.
The Menu file is empty.
Warning: The following Program file could not be found: *file name*
Warnings were encountered converting the Timed Start Menu. Press Exit to Continue.

Running the AR Program

After writing a TSM, you can run the program by completing the following procedures.

- Creating the *AR.linkcommand* file (see below).
- Creating the *MAKE_AR.text* file (see below).
- Creating the *AR.program* file.
- Using DataCommC to run the program.

AR.linkcommand File.

The ASCII link command file (see figure 2-2) defines the paths to each object file called by *MAKE_AR.text*.

```
c:C/HP/AR_MENU.object  
c:C/HP/AR_RUN.object  
c:C/HP/AR_UI.object
```

Figure 2-2. AR.linkcommand File

Creating the Program File

You create the program file under two conditions - (1) prior to running the program for the first time, and (2) whenever you modify a source file. The *MAKE_AR.text* file (see figure 2-1) compiles the program creating object files, makes the linkages between the object files and generates the *AR.program* file.

Whenever you modify any source code *MAKE_AR.text* compiles only the source files that have changed.

The format for this file is as follows.

- All entries start in column 1.
- Entries starting with a pound sign (#) are comments.
- Lines ending in a backslash (\) are continued on the following line.
- Lines starting with the dollar sign (\$) describe the sequence of execution for creating the program.

For a complete description of *MAKEFILE.text* files see the *DataCommC User's Guide*.

```
# Make file for Automatically Running Programs.
#

OBJ=AR_MENU.OBJECT AR_RUN.OBJECT AR_UI.OBJECT

PROGRAM=AR

$(PROGRAM).program: $(OBJ)
    link $(PROGRAM).linkcommand
```

Figure 2-3. MAKE_AR.text File

Auto Run Menu Source File

This file gets the TSM and translates it into the data base found in *current_menu*. The major routines of this program are as follows.

s_type

As each TSM line is parsed, this routine returns the line type, the TSM key character. It is passed a pointer to the string and a pointer to an index into the string. Possible return values are FILENAME (f), TIME (t), DATE (d), and EVERYDAY (de), respectively. All other characters returned are classed as OTHER.

get_filename

This routine obtains the file name from the TSM program line. It is passed a pointer to the string containing the file name, a pointer to an index into the file name, and a pointer to a character array where it will place the file name. It returns FALSE if successful else it returns TRUE.

getval

This routine generates a decimal number from a numeric ASCII string. It is used to obtain hours, minutes, year, month and day information. It is passed a pointer to a string, a pointer to an index into the string, an address to a return variable, a minimum and maximum value to accept, and an expected termination character. It returns FALSE if successful else it returns TRUE.

gettime

This routine obtains the time information from the TSM time line. It is passed a pointer to the string, a pointer to an index into the string, and pointers to the variables where the hours and minutes are to be placed. It returns FALSE if it is successful, else it returns TRUE.

getdate

This routine obtains the date information from the TSM date line. It is passed a pointer to the string, a pointer to an index into the string, and pointers to the variables where the day, month and year are to be placed. It returns FALSE if it is successful, else it returns TRUE.

convert_menu

This routine oversees the conversion of the TSM into the data base in *current_menu*. It is passed a string containing the name of the current TSM and a FILE structure for the file containing the TSM. WARN is returned if only non fatal warnings are encountered, ERROR is returned if any fatal errors are encountered else FALSE is returned.

read_menu

This routine is called from the top level User Interface menu to select a TSM. It prompts the user for the name of the TSM and searches for the menu. If the TSM is found, it starts the conversion. If all the steps are successful, FALSE is returned. If the menu cannot be converted or if the user enters QUIT, TRUE is returned.

init_menu

This routine initializes the data base in *current_menu*.

AR_MENU.csource File Listing

```

/*****
*
*          HP 18320A DataCommC
*      Automatically Running a Program
*
*          File: AR_MENU.csource
*
*          Copyright 1989
*      Hewlett-Packard Company
*
*  This program selects the TSM and translates it into "current_menu."
*****/

#include "c:C/HP/AR_H.include"

/*****
*  The "s_type" routine returns the TSM line type.
*
*  PARAMETERS:
*      s - Address of the null-terminated character array holding the line being
*          parsed.
*
*      p - Address of the current index to s. The index will be 0 entering this
*          routine. The index will be updated to select the character after the line
*          specifier.
*          If a line specifier is not found then the index will be unaffected.
*
*  RETURNS:
*      TIME      : The line type "t."
*      DATE      : The line type "d."
*      FILENAME  : The line type "f."
*      EVERYDAY  : The line type "de."
*      OTHER     : Any other line type value.
*****/

s_type(s, p)
char s[];
int *p;

{
    /* Determine the string type. */
    if (((s[*p] == 'F') || (s[*p] == 'f')) && (s[(*p)+1] == ' ')) /* Program string. */
    {
        (*p)++;
        return(FILENAME);          /* File name string. */
    }
    if ((s[*p] == 'D') || (s[*p] == 'd')) /* Date string. */
    {
        if (s[(*p)+1] == ' ')
        {
            (*p)++;
            return(DATE);
        }
        (((s[(*p)+1] == 'E') || (s[(*p)+1] == 'e')) && /* Every day string. */
        ((s[(*p)+2] == ' ') || (s[(*p)+2] == ';'))))
    }
}

```

AR_MENU.csource File Listing (Continued)

```
    {
        *p = *p + 2;
        return (EVERYDAY);
    }
    if (((s[*p] == 'T') || (s[*p] == 't')) && (s[(*)+1] == ' ')) /* Time string. */
    {
        (*p)++;
        return (TIME);
    }
    return(OTHER); /* Other string. */
}
```

AR_MENU.csource File Listing (Continued)

```

/*****
* The "get_filename" routine gets the TSM file name.
*
* PARAMETERS:
*   s - Address of the null-terminated character array holding the line being
*       parsed.
*   p - Address of the current index to s. On entering this routine, the index
*       will select the space following the file name specifier. During this
*       routine the index will be updated to select the character after the
*       file name.
*       If the routine is successful this character will be a ";".
*       If the routine is not successful, the index will point to a ' ',
*       a '0\', a carriage return, or a CNTL Z.
*   fname - A pointer to a null-terminated string which receive the TM file name.
*
* RETURNS:
*   FALSE - Successful
*   TRUE - Unsuccessful
*****/
get_filename(s,p,fname)
char s[];
int *p;
char fname[];

{
    int i;

    for (; ((s[*p] != '\0') && (s[*p] == ' ')); (*p)++);

    if (s[*p] == '\0')
        return (TRUE);
    for (i=0; ((s[*p] != '\0') && (s[*p] != '\015') &&
        (s[*p] != '\032') && (s[*p] != ' ') &&
        (s[*p] != ';'))); i++, (*p)++)
        fname[i] = s[*p];

    fname[i] = '\0';

    if (strlen(fname) < 1)
        return (TRUE);
    if (s[*p] != ';')
        return (TRUE);

    return(FALSE);
}

```

AR_MENU.csource File Listing (Continued)

```

/*****
* The "get_val" routine generates a decimal number from a numeric ASCII string that
* defines the hours, minutes, day, month, and year information.
*
* PARAMETERS:
*   s      - Address of the null-terminated character array holding the line being
*             parsed.
*   p      - Address of the current index to s. On entering this routine, the index
*             will select the first ASCII number to be converted or to a preceding
*             space. During this routine the index will be updated to select the
*             character after the consecutive ASCII numeric character in the string
*             or to the last space following the last ASCII numeric character in the
*             string. If successful this will be the terminating character.
*   v      - Address of the variable in which to put the result.
*   term   - The terminating character.
*   min    - The minimum acceptable value.
*   max    - The maximum acceptable value.
*
* RETURNS:
*   FALSE - Successful
*   TRUE  - Unsuccessful
*****/

getval(s, p, v, term, min, max)
char s[];
int *p, *v;
int term;
int min, max;

{
    int t;

    for (; ((s[*p] != '\0') && (s[*p] == ' ')); (*p)++);
    if (s[*p] == '\0')
        return(TRUE);
    t = 0;
    while((s[*p] >= '0') && (s[*p] <= '9'))
    {
        t = (t * 10) + (s[*p] - '0');
        (*p)++;
    }
    if ((t < min) || (t > max))
        return(TRUE);

    for (; s[*p] == ' '; (*p)++);

    if(s[*p] != term)
        return(TRUE);
    *v = t;
    return(FALSE);
}

```

AR_MENU.csource File Listing (Continued)

```

/*****
*   The "get_time" routine obtains the time information from a time line in a TSM
*
*   PARAMETERS:
*
*       s       - Address of the null-terminated character array holding the line being
*                 parsed.
*       p       - Address of the current index to s. On entering this routine, the index
*                 will select the space following the time line specifier. During this
*                 routine the index is updated by the getval function.
*       hour    - The address of the variable where the hour is stored.
*       min     - The address of the variable where the minute is stored.
*
*   RETURNS:
*       FALSE - Successful
*       TRUE  - Unsuccessful
*****/

gettime(s,p,hour,min)
char s[];
int *p, *hour, *min;

{
    int h, m;

    if (getval(s, p, &h, ':', 0, 23))
        return(TRUE);

    (*p)++;

    if (getval(s, p, &m, ';', 0, 59))
        return(TRUE);
    *hour = h;
    *min = m;
    return(FALSE);
}

```

AR_MENU.csource File Listing (Continued)

```

/*****
* The "getdate" routine gets the date information from the date line in a TSM.
*
* PARAMETERS:
*   s - Address of the null-terminated character array holding the line being
*       parsed.
*   p - Address of the current index to s. On entering this routine, the index
*       will select the space following the date line specifier. During this
*       routine the index is updated by the getval routine.
*   day - Address of the variable where the day is stored.
*   month - Address of the variable where the month is stored.
*   year - Address of the variable where the year is stored.
*
* RETURNS:
*   FALSE - Successful
*   TRUE - Unsuccessful
*****/

getdate(s,p,day,month,year)
char s[];
int *p, *day, *month, *year;

{
    int d, m, y;
    if (getval(s, p, &m, '/', 1, 12))
        return(TRUE);

    (*p)++;
    if (getval(s, p, &d, '/', 1, 31))
        return(TRUE);

    (*p)++;
    if (getval(s, p, &y, ';', 88, 99))
        return(TRUE);

    if ((m == 9) || (m == 4) || (m == 6) || (m == 11)) && (d > 30))
        return(TRUE); /* Only 30 days in these months. */
    if ((m == 2) && (d > 29))
        return(TRUE); /* February has 29 days in a leap year. */

    if ((m == 2) && ((y % 4) != 0) && (d > 28))
        return (TRUE); /* February has 28 days in non-leap years. */
    *day = d;
    *month = m;
    *year = y;
    return(FALSE);
}

```

AR_MENU.csource File Listing (Continued)

```

/*****
*   The "convert_menu" routine coverts TSM data into "current_menu."
*
*
*   PARAMETERS:
*       fname - Address of the null-terminated character array containing the name of
*               the TSM.
*       fid   - Address of the structure variable for accessing files.
*
*   RETURNS:
*       FALSE - No errors or warnings.
*       WARN  - Warnings and no errors.
*       ERRORS - Errors.
*****/

convert_menu(fname,fid)
char *fname;
FILE *fid;

{
    int date_index, time_index, file_index, min, hour, day, month, year;
    char *header = "<tmf>";
    char s[MAX_STR_LEN];
    int error = FALSE, warning = FALSE;
    char tfname[MAX_STR_LEN];
    int p, i;
    FILE *tfid;

    printf("Building the data base from the Menu file.\n    %s\n\n",fname);

    if (fgets(&s,MAX_STR_LEN,fid) == NULL) /* Get the first line to parse. */
    {
        printf("The Menu file is empty.\n");
        return(ERROR);
    }

    for (i=0; i<strlen(header); i++) /* Convert uppercase character to lowercase. */
        s[i] = tolower(s[i]);

    if (strncmp(&s,header,strlen(header)) != 0)
    {
        printf("The Menu file did not start with    %s\n", header);
        return(ERROR);
    }

    file_index = date_index = time_index = 0;
    while ((fgets(&s, MAX_STR_LEN, fid) != NULL) && (error != TRUE))
    {
        /* Get the next line to parse. */
        p=0;
        switch (s_type(&s, &p))
        {
            case FILENAME:
                if ((file_index != 0) && ((time_index == 0) || (date_index == 0)))

```

AR_MENU.csource File Listing (Continued)

```
{
    printf("Error: Either no start date or time was specified for the
           Program      file:\n %s\n",tfname);
    error = TRUE;
    break;
}
if (time_index != date_index)
{
    printf("Error: A mismatched number of start times and dates were
           specified for the\n      Program file: %s\n",tfname);
    error = TRUE;
    break;
}
if (file_index >= NUM_ENTRIES)
{
    printf ("Error: More than %d Program files were specified.
            \n",NUM_ENTRIES);
    error = TRUE;
    break;
}

if (!get_filename(&s,&p,&tfname))
{
    strcpy(current_menu.user_time_entry[file_index].user_app,tfname);
    if ((tfid = fopen(tfname, "r")) == NULL)
    {
        warning = TRUE;
        printf("Warning: The following Program file could not be found:
               \n                %s\n", tfname);
    }
    else
        fclose(tfid);
    current_menu.user_time_entry[file_index].entry_active = TRUE;
    file_index++;
    date_index = time_index = 0;
}
else
{
    printf("Error: A program file name was incorrectly specified:
           \n      %s\n", tfname);
    error = TRUE;
}
break;

case TIME:
    if (file_index == 0)
    {
        printf("Error: A start time was specified before any Program files.");
        error = TRUE;
        break;
    }
    if (time_index >= NUM_TIMES)
```

AR_MENU.csource File Listing (Continued)

```
{
    printf("Error: Too many start times were specified for Program file
           \n      %s\n",tfname);
    error = TRUE;
    break;
}

if (gettime(&s,&p,&hour,&min))
{
    printf("Error: A time was incorrectly specified for Program file
           \n      %s\n",tfname);
    error = TRUE;
    break;
}
current_menu.user_time_entry[file_index-1].user_times_active[time_index]
    = TRUE;
current_menu.user_time_entry[file_index-1].user_times[time_index].tm_hour
    = hour;
current_menu.user_time_entry[file_index-1].user_times[time_index].tm_min
    = min;
time_index++;
break;

case EVERYDAY:
    if (file_index == 0)
    {
        printf("Error: A start date was specified before any Program files.");
        error = TRUE;
        break;
    }
    if (date_index >= NUM_TIMES)
    {
        printf("Error: Too many start dates were specified for Program file
               \n      %s\n",tfname);
        error = TRUE;
        break;
    }
    current_menu.user_time_entry[file_index-1].everyday[date_index] = TRUE;
    date_index++;
    break;
case DATE:
    if (file_index == 0)
    {
        printf("Error: A start date was specified before any Program files.");
        error = TRUE;
        break;
    }
    if (date_index >= NUM_TIMES)
    {
        printf("Error: Too many start dates were specified for Program file:
               \n      %s\n",tfname);
        error = TRUE;
        break;
    }
}
```

AR_MENU.csource File Listing (Continued)

```
    if (getdate(&s,&p,&day,&month,&year))
    {
        printf("Error: A date was incorrectly specified for Program file
               \n      %s\n",tfname);
        error = TRUE;
        break;
    }

    current_menu.user_time_entry[file_index-1].user_times[date_index].tm_day
        = day;
    current_menu.user_time_entry[file_index-1].user_times[date_index].tm_mon
        = month;
    current_menu.user_time_entry[file_index-1].user_times[date_index].tm_year
        = year;
    date_index++;
    break;

case OTHER:
    break;
default:
    break;
}
}
```

AR_MENU.csource File Listing (Continued)

```

/*****
* The program reaches this point either at the end of the TSM or when an error occurs.*
*****/

    if ((!error) && (file_index == 0))
    {
        printf("Error: No Program files were specified.\n");
        error = TRUE;
    }
    if ((!error) && ((time_index == 0) || (date_index == 0)))
    {
        printf("Error: Either no start date or time was specified for the Program file:
            \n      %s\n",tfname);
        error = TRUE;
    }

    if ((!error) && (time_index != date_index))
    {
        printf("Error: A mismatched number of start times and dates were specified for the
            \n      Program file: %s\n",tfname);
        error = TRUE;
    }

    if (error)
    {
        init_menu();
        return(ERROR);
    }
    if (warning)
        return(WARN);

    return(FALSE);
}

```

```

/*****
* The "read_menu" routine calls the top level user interface menu for selecting a      *
* TSM.                                                                                   *
*                                                                                       *
* PARAMETERS:                                                                           *
*   fname - Address of the null-terminated character array containing the name of      *
*   the TSM.                                                                           *
*                                                                                       *
* RETURNS:                                                                              *
*   FALSE - Menu could be read and there were no errors when building the data      *
*   base.                                                                               *
*   TRUE  - Menu could not be read or there were errors when building the data      *
*   base.                                                                               *
*****/

```

```

read_menu(fname)
char fname[];

```

```

{

```

AR_MENU.csource File Listing (Continued)

```
FILE *fid;
int c, tr;

init_menu();
while (TRUE)
{
    wipe(1,25);
    set_cursor(10,1);
    printf("Enter the name of the Timed Start Menu - ");
    scanf("%40s",fname);
    set_cursor(12,1);
    if ((fid = fopen(fname, "r")) == NULL)
    {
        set_cursor(14,1);
        printf("The Menu file could not be opened - Try again or Exit",20);
        put_sk("Try Again",SK7,INVERSE);
        put_sk("Exit",SK8,INVERSE);
        while ((c = getch()) != SK7) && (c != SK8));
        if (c == SK8)
        {
            fname[0] = '\0';
            return(TRUE);
        }
    }
    else
    {
        wipe(1,25);
        set_cursor(1,1);
        if ((tr = convert_menu(fname, fid)) == ERROR)
        {
            fclose(fid);
            set_cursor(21,1);
            printf("Errors were encountered converting the Timed Start Menu.\n");
            printf("Press Exit To Continue");
            put_sk("Exit",SK8,INVERSE);
            while ((c = getch()) != SK8);
            return (TRUE);
        }
        if (tr == WARN)
        {
            fclose(fid);
            set_cursor(21,1);
            printf("Warnings were encountered converting the Timed Start Menu.\n");
            printf("Press Exit To Continue");
            put_sk("Exit",SK8,INVERSE);
            while ((c = getch()) != SK8);
            return (FALSE);
        }
        fclose(fid);
        return(FALSE);
    }
}
return(FALSE);
}
```

AR_MENU.csource File Listing (Continued)

```
/******  
* The "init_menu" routine initializes the "current_menu" data base. *  
*****/  
  
init_menu()  
{  
    int i,j;  
    for(i=0;i<NUM_ENTRIES;i++)  
    {  
        current_menu.user_time_entry[i].entry_active = FALSE;  
        current_menu.user_time_entry[i].user_app[0] = '\0';  
        for(j=0;j<NUM_TIMES;j++)  
        {  
            current_menu.user_time_entry[i].user_times_active[j] = FALSE;  
            current_menu.user_time_entry[i].user_times[j].tm_msecs = 0;  
            current_menu.user_time_entry[i].user_times[j].tm_sec = 0;  
            current_menu.user_time_entry[i].user_times[j].tm_min = 0;  
            current_menu.user_time_entry[i].user_times[j].tm_hour = 0;  
            current_menu.user_time_entry[i].user_times[j].tm_day = 1;  
            current_menu.user_time_entry[i].user_times[j].tm_mon = 1;  
            current_menu.user_time_entry[i].user_times[j].tm_year = 89;  
            current_menu.user_time_entry[i].everyday[j] = FALSE;  
        }  
    }  
}
```

Auto Run Timing File

The three routines in this file check the real time clock and initiates (spawns) program files at the appropriate time **and on the proper date**. They works off a the menu data base *current_menu*.

same_date

Determines if the dates (day, month and year) are the same in two type tm structures.

same_time

Determines if the times (hours and minutes) are the same in two type tm structures.

run_menu

Every 10 seconds this routine determines if a analyzer clock time minute has incremented. If it has, the data base *current_menu* is searched for the program file to run.

AR_RUN.csource File Listing

```

/*****
*
*          HP 18320A DataCommC
*      Automatically Running Programs
*
*          File: AR_RUN.csource
*
*          Copyright 1989
*      Hewlett-Packard Company
*
* This file contains the code which checks the real time clock and initiates (spawns)
* the menu program files at the appropriate time and on the proper date. It works
* off data base "current_menu."
*****/

#include "c:C/HP/AR_H.include"

/*****
* The "same_date" routine determines if the date in the TSM structure and the real
* time clock structure are the same.
*
* PARAMETERS:
*   ct - Current time pointer.
*   lt - Last time pointer.
*
* RETURNS:
*   TRUE : The day, month, and year in both structures are the same.
*   FALSE : One or more values in the two structures do not match.
*****/
same_date(ct, lt)
struct tm *ct, *lt
{ if ((ct->tm_mon == lt->tm_mon) &&
    (ct->tm_day == lt->tm_day) &&
    (ct->tm_year == lt->tm_year)) return(TRUE);
    return(FALSE);
}

```

AR_RUN.csource File Listing (Continued)

```

/*****
* The "same_time" routine determines if the time in the TSM and real time clock are
* the identical.
*
* PARAMETERS:
*   ct - Current time pointer.
*   lt - Last time pointer.
*
* RETURNS:
*   TRUE : The hours and minutes in both structures are the same.
*   FALSE : Both values in the structures do not match.
*****/

same_time(ct, lt)      /* Current time pointer. */
struct tm *ct, *lt;    /* Last time pointer. */
{
    if ((ct->tm_min == lt->tm_min) &&
        (ct->tm_hour == lt->tm_hour))
        return(TRUE);
    return(FALSE);
}

/*****
* The "run_menu" routine determines when a minute has passed and then uses the
* "start_time" routine. This routine tests this status every 10 seconds.
*
* RETURNS:
*   NONE
*****/

run_menu()      /* No return for this function. */
{
    static struct tm current_time, last_time;
    int first_pass, check, i, j;
    long lp;

    first_pass = TRUE;      /* First time through the while loop. */

    while(TRUE)
    {
        wait(1001);          /* Wait for 10 seconds. */
        get_time(&current_time); /* Get the real time clock value. */
        if (first_pass)
        {
            check = TRUE;      /* Force a pass through the data base. */
            first_pass = FALSE; /* Second and subsequent passes through data base. */
        }
        else /* Make a pass through the data base when the hour and minutes have *
            * changed.*/
            check = !same_time(&current_time, &last_time);
        if (check)
        {
            for (i=0; i<NUM_ENTRIES; i++) /* Cycle through all the menu program files. */

```

AR_RUN.csource File Listing (Continued)

```
{
    for (j=0;j<NUM_TIMES;j++) /* Cycle through all the menu start times. */
    {
        if (((current_menu.user_time_entry[i].entry_active) &&
            current_menu.user_time_entry[i].user_times_active[j]) &&
            same_time(&current_time,
                &current_menu.user_time_entry[i].user_times[j])) &&
            (same_date(&current_time,
                &current_menu.user_time_entry[i].user_times[j])
                || current_menu.user_time_entry[i].everyday[j]))
        {
            if (spawn(current_menu.user_time_entry[i].user_app,
                0,
                01,
                100,
                &lp))
                printf("\nCould not spawn program:\n    %s\n",
                    current_menu.user_time_entry[i].user_app);
        }
    }
}
last_time = current_time;
}
```

Auto Run User Interface File

This file only contains the top level User Interface code. It permits the user to load a TSM, run a TSM if one has been successfully loaded or to exit the program.

AR_UI.csource File Listing

```
/*
*****
*
*          HP 18320A DataCommC
*          Automatically Running Programs
*
*          File: AR_UI.csource
*
*          Copyright 1989
*          Hewlett-Packard Company
*
*
* This file contains the top level user interface code for loading a TSM, running a
* TSM, or exiting the program.
*****
*/

#include "c:C/HP/AR_H.include"

main()
{
    int terminate_reason = CONTINUE;    /* Controls 1st level UI menu exits. */

    int valid_menu = FALSE;    /* Valid menu loaded indicator. */

    char menu_filename[MAX_STR_LEN];    /* Current menu name. */
    cursor_on();
    menu_filename[0] = '\0';
    wipe(1,25);

    while (terminate_reason == CONTINUE)
    {
        wipe(1,25);
        put_all_sks("Load\TS Menu",
                    (valid_menu == TRUE) ? "Run\TS Menu" : "", "", "", "", "",
                    "\Exit", INVERSE);

        set_cursor(1,5);
        if (valid_menu == FALSE)
            printf("No Timed Start Menu is currently loaded.\n\n\n");
        else
            printf("Currently loaded Timed Start Menu:  %s\n\n\n", menu_filename);
        printf("Load TS Menu - Load a Timed Start Menu from disc.\n\n");
        if (valid_menu == TRUE)
            printf("Run TS Menu - Run the currently loaded timed start menu.\n\n");
    }
}
```

AR_UI.csource File Listing (Continued)

```
else
    printf("\n\n");

printf("Exit      - Exit this program.\n\n\n");
printf("          MAKE A SELECTION VIA A SOFTKEY");

terminate_reason = CONTINUE;
switch (getch())    /* Convert valid menu switch. */
{
    case SK1:
        valid_menu = !read_menu(&menu_filename);
        break;
    case SK2:
        if (valid_menu == TRUE) terminate_reason = RUN;
        break;
    case SK8:
        terminate_reason = DONE;
        break;
    default:
        break;
}

if(terminate_reason == RUN)
{
    /* Execute the Timed Start Menu. */
    wipe(1,25);
    set_cursor(1,1);
    cursor_off();
    keyboard_unlock();
    run_menu();
}
}
```

Auto Run Include File

This include file defines the following.

- The DataCommC include files required for auto run operations.
- The maximum number of program files that can be specified (NUM_ENTRIES).
- The maximum number of start times and start dates that can be specified (NUM_TIMES).

You can change the number of program files in each TSM or the times each program can run by changing NUM_ENTRIES or NUM_TIMES as appropriate.

AR_H.include File Listing

```
/*
 *
 *      HP 18320A DataCommC
 *      Automatically Running a Program
 *
 *      File: AR_H.include
 *
 *      Copyright 1989
 *      Hewlett-Packard Company
 *
 *      This file contains the DataCommC include file references and defines AR program
 *      parameters.
 */
#include <time.include>
#include <video.include>
#include <conio.include>
#include <stdio.include>
#include <system.include>

#define MAX_STR_LEN 81      /* The maximum length of name strings. */
/*
 * NUM_TIMES is the maximum number of start times and dates for EACH program file.
 *
 * NUM_ENTRIES is the maximum number of program files per TSM.
 *
 * You can change either of these values to meet your requirements. After changing
 * them compile all files.
 */
#define NUM_TIMES 5      /* Maximum start times and dates per program file. */
#define NUM_ENTRIES 5    /* Maximum number of program files. */
```

AR_H.include File Listing (Continued)

```
/* Function Return Values. */
#define CONTINUE 0
#define RUN 1
#define DONE 2
#define FILENAME 0
#define TIME 1#define EVERYDAY 2
#define DATE 3
#define OTHER 4

#define ERROR 2
#define WARN 1

/* TSM Structures and Variables */

struct basic_entry
{
    int entry_active;
    char user_app[MAX_STR_LEN];
    short user_times_active[NUM_TIMES];
    struct tm user_times[NUM_TIMES];
    int everyday[NUM_TIMES];
};

struct menu_struct
{
    struct basic_entry user_time_entry[NUM_ENTRIES];
} current_menu;
```

AR_TEST.csource File Listing

To use this file to test a new TSM involves using the DataCommC *Make* function to create *AR_TEST.program*. You then Insert *AR_TEST.program* into your TSM and run the *AR.program*. The *AR.program* then prints the program name and a time stamp on the first blank screen row.

AR_TEST.csource File Listing

```

/*****
*
*      HP 18320A DataCommC
*
*      File: AR_TEST.csource
*      REVISION A.00.00
*
*      Copyright 1989
*      Hewlett-Packard Company
*
*      This file tests TSMS.
*****/

#include <time.include>

main(argc,argv)
int argc;
char *argv[];
{
    struct tm td;
    int i, j, r, a;
    char c;

    get_time(&td);                /* Get the current time and date. */

    for (r=1; r<=20; r++)        /* Search for the first blank row. */
    {
        read_vidram(r,1,&c,&a);
        if ( c == ' ')
            break;
    }
    if (r == 21)                 /* No blank rows available. */
    {
        r = 20;
        for (i=1; i < 20; i++)
        {
            for (j=1; j<=80; j++) /* Scroll the display. */
            {
                read_vidram(i+1,j,&c,&a);
                write_vidram(i,j,c,a);
            }
        }
        wipe (20,20)
    }
    set_cursor(r,1);
    printf("Program = %s   Time = %2.2u:%2.2u:%2.2u   Date = %2.2u/%2.2u/%2.2u",
        argv[0],
        td.tm_hour, td.tm_min, td.tm_sec
        td.tm_mon, td.tm_day, td.tm_year);
}

```


Decoding Displays

DataCommC has a BOPS DECODE (Bit Oriented ProtocolS) program for examining and displaying frame data in general. With this solution program you can develop a BOPS DECODE program tailored to your requirements. This solution program uses the *DataCommC Decode Display Kernel* that works with the DataCommC platform to handle both Examine Buffer and the Run Time displays.

The display kernel eliminates the requirements for programming routine operations such as:

- Getting softkeys.
- Managing the screen while rolling, paging, or jumping.
- Working differently for Examine Data and Run Time displays.

The only thing you must provide is a routine that decodes one event and writes to a *virtual screen* using a special routine `dispf( )`. A structure that defines the softkeys that control the decode format is also required. This chapter covers these items in detail.

Solutions Disc 1 contains the source code corresponding to the BOPS DECODE program. The include files and the display kernel object files for linking are also on the Solutions disc. The user can either modify these files to change the decode or use them as an example for writing a decode.

Display Decoding Files

The files used by BOPS DECODE, and their general functions are as follows.

BOPS.linkcommand	Defines the <i>object</i> files to link.
DECODE.include	Defines key style values and the decode keys structure.
DSDISP.object	A display kernel object file.
DSEXAM.object	A display kernel object file.
DSMAIN.object	A display kernel object file.
DSRTD.object	A display kernel object file.
GBOPS.csource	Extracts frame and protocol information from an event.
GBOPS.include	Defines constants and structures for decoding frames and protocols.
GBOPS.text	The "MAKEFILE" that creates the "BOPS.program" file (see "Creating the Program File" in the "Running BOPS Decode" section).
GBOPS_DEC.csource	Displays the information extracted from the event by GBOPS.csource file.
GBOPS_KEYS.csource	Defines the display decoding softkeys.
GBOPS_LEAD.csource	Displays lead changes and lead status.
GLOBUS.include	Defines data type, softkey constants, and display contents.
MACROS.include	Contains the definitions for Raw Bytes and Undecoded .
MAIN_DEC.csource	The main routine for decoding and displaying a single event.
USER.csource	Creates the screen header and passes the user decode routine location to the display kernel.

Creating a Display Decode Program

A display decode program such as the sample decoding program (see figure 3-1) requires the following four items. (The elements used in Figure 3-1 program are shown in parentheses.)

1. A data structure (*format_keys*) that defines the softkey tree.
2. A routine (*decode*) that decodes one event.
3. A string (*screen_heading*) that contains the screen heading.
4. A **user()** routine that calls the **display()** routine. The **display()** routine parameters are *format_keys*, *decode*, and *screen_heading*, the first three items..

When writing a display file, you would normally have one file for the softkey structure, one file for *user* and the screen heading, and possibly several decode routines. A *main* file that contains the routine passed to **display()**, and subsidiary files containing routines called by *main*. The only restriction on naming files and structures is that the **user()** routine must call the **display()** routine. The following sample decoding program has all four items in a single file.

```
#include "message.include"      /* EVENT */
#include "DECODE.include"      /* DECODE_KEYS, CYCLE, OFFON, LAYER, BLANK, NIL */

extern void dispf( );
extern void display( );

/*****
 *
 *          ITEM 1
 *   Your definition of the "Display Format" softkeys.
 *   The BOPS decode uses GBOPS_KEYS.csource to define softkeys.
 *****/
#include "DECODE.include"
#include "MACROS.INCLUDE"

DECODE_KEYS format_keys[7] = {
    {OFFON, {"Event #"}, {1}},
    {CYCLE, {"Hex", "ASCII", "EBCDIC"}, {1}},
    {OFFON, {"Raw Bytes"}, {1}};

/*****
 *
 *          ITEM 2
 *   The "decode( )" routine accepts the parameters "event" and "event_num" to decode
 *   one event.
 *   The BOPS decode uses MAIN_DEC.csource to display an event.
 *****/

void decode(event, event_num)
EVENT *event;
int event_num;
{
    if(format_keys[0].value)
        dispf("\nEvent %5d: ", event_num);
}

/*****
 *
 *          ITEM 3
 *   The "screen_heading" is the character string that contains your screen heading.
 *****/
```

Figure 3-1. Sample Decoding Program

```

*                                     ITEM 4                                     *
*   The "user( )" routine passes the location of the softkey definition, the decode *
*   routine, and the screen heading to the display kernel.                       *
*   *                                                                           *
*   The routine name MUST BE user and it must call the routine "display( )".    *
*   *                                                                           *
*   The "display( )" routine MUST HAVE the parameters "format_keys," "decode", and *
*   "screen_heading."                                                            *
*   *                                                                           *
*   For GBOPS_DEC.csource this file is USER.csource.                          *
*****/

void user ( )
{
    char *screen_heading =
    "          TE                                     NT          ";
/* 1234567890123456789012345678901234567890123456789012345678901234567890 */
/*          Use this template to align your screen headings.                */

    display(format_keys,decode,screen_heading);
}

```

Figure 3-1. Sample Decoding Program (Continued)

BOPS DECODE Decoding Structure

The MAIN_DEC.csource file contains the primary decode routine **main_dec()** that has the following functions.

- Call the subroutine (**decode_frame()**) to parse the information for display.
- Display the frame information (**display_frame()**)
- Display the lead state (**lead_stat_disp()**).
- Display the lead changes (**lead_change_disp()**).
- Display raw Level 3 data.
- Display the event number and time stamp.

decode_frame()

This routine in GBOPS.csource returns a structure defined in GBOPS.include. The information returned in the structure includes such things as address, N(r), N(s), etc.

display_frame()

This routine in GBOPS_DEC.csource uses the structure returned by **decode_frame()** and outputs the information to the display.

Modifying Frame Decoding

Changing `decode_frame( )` routine in the `GBOPS.csource` file is one way you can change the decoding of a portion of the frame. To do this, change the information passed to the `DEC_GBOPS` structure (*out_ptr* points to this structure). Then create a *newBOPS.program* with the `GBOPS.text` file.

Adding Frame Information

Decoding and displaying additional information involves the following changes.

- Modifying the `DEC_GBOPS` structure in `GBOPS.include` file.
- Modifying the `decode_frame( )` routine in the `GBOPS.csource` file.
- Modifying the `display_frame( )` routine in the `GBOPS_DEC.csource` file.

To do this:

1. Add a field to the `DEC_GBOPS` structure.
2. Modify the `decode_frame( )` routine to fill in this new value.
3. Modify `display_frame( )` to display the *new structure field*.
4. Compile the program using the `GBOPS.text` file.

The Formatted Display Routine

The display kernel **dispf()** routine is similar to the **printf()** routine in concept and execution. However, **dispf()** is used to display decoded data on the screen. For a description of this routine see "Figure 3-2. **dispf()** Routine Description."

Rules

Using the **dispf()** routine requires you to observe the rules defined in the following table.

Table 3-1. Rules for Using the **dispf() Routine**

1. There is a *maximum* of 80 characters per line. Longer lines are truncated. There is *no wraparound* as in the **printf()** routine.
2. There can be up to 1,024 lines *per event*, defined with the line feed character `\n`.
3. The **dispf()** routine *starts* with `\n`. There is no line for writing when the decode routine takes control, therefore you must create one. Ending **dispf()** with `\n` places a blank line after the display and is *not recommended*.
4. *The only backslash character supported by **dispf()** is `\n`.*
5. Any columns on a line *not specified* are blank and use the *default* attributes. This eliminates the requirement for clearing part of the screen when doing two column displays.
6. Attributes are only applied to the field you specify. For four successive fields to use inverse video, the *I* attribute must be set for each field. If you only specify the *I* attribute for the first field, the other three will be in normal video.

The following table describes the **dispf()** routine.

Table 3-2. dispf() Routine Description

Syntax	Items in parentheses () are optional. Items preceded by an n can be repeated from 0 to an infinite number of times. The number of repetitions is memory dependent. Items in brackets []: choose one.
Format	<code>dispf(format_string (, arglist))</code> <code>format_string = n(n(pc) + n(format_specification))</code> <code>format_specification = % + n(Attribute) + (Justification) + (Width) + Type</code> <code>pc = [printable character> \n]</code>
Attribute Parameters	Attribute = [B H I X E U] where: B = blinking X = hexadecimal H = half-bright E = EBCDIC I = inverse video U = underline default: non-blinking normal video full-bright ascii
Justification Parameters	Justification = [0 -] where: 0 = right justify, padded with zeros. - = left justify default: right justify
Width Parameters	Width = [<numeral> *] where: * = width was passed as a parameter.
Type Parameters	Type = [d ld x lx s r p] where: d = decimal ld = long decimal x = hexadecimal lx = long hexadecimal s = null-terminated string r = raw string (i.e., NOT null-terminated) p = position cursor to column (1 - 80) default: d, ld, x, lx, s: min field required r: 80 p: 1 (i.e., column 1)

Examples

The following examples demonstrate using `dispf( )` to create special displays.

Example 1 - Time Stamp

This `dispf( )` call places a time stamp in columns 29 - 40 in the format "hh:mm:ss.msc." To ensure there are no blanks in the time stamp (e.g., 12: 3: 9.789) *zero pad justification* is used.

```
dispf("%29p%02d:%02d:%02d.%03d",hour,minute,second,msec);
```

Example 2 - "Cramped Hex" Format

The "cramped hex" format places two hexadecimal digits per character cell. The following `dispf( )` call will enter a frame I-field in this format, using inverse video starting in column 2. The I-field will be truncated after the first 79 characters.

```
dispf("%2p%IX*r",i_field_length,i_field_start_ptr);
```

Example 3 - 10 Lines of Blinking, Centered, Inverse Video

The following `dispf( )` call displays 10 lines of "Hello" in blinking, inverse video in the middle screen.

```
for(i = 0; i < 10; i++)  
dispf("\n%38;%BIs","Hello");
```

Example 4 - Event Information in Column 20

The following `dispf( )` call displays I-frame information N(r) and N(s) at column 20.

```
dispf(""%20pnr: %2x ns: %2x",nr,ns);
```

Modifying the Display

To use a different display format involves editing the `MAIN_DEC.csource` file or the `GBOPS_DEC.csource` file `dispf( )` formats using the rules and procedures discussed above.

The Softkey Structure

A decode routine reads the `DECODE_KEYS` structure *value* field to determine how to decode and display information. The softkey types and their values are described below.

Note

References to *first label* mean `label[0]`.
Define *only softkeys 1 through 7*. Softkey 8 is reserved by the display kernel as the **Exit** key.

BLANK Type Softkey

The softkey is displayed in full-bright video with no characters. Therefore, the labels are ignored; the value is never updated and remains at 0.

The following softkey structure would:

- Set softkeys 1 and 3 to BLANK.
- Set softkeys 2 and 4 to OFFON.
- Softkeys 5, 6, and 7 would also be BLANK because all unspecified parts of the structure are initialized to 0.

```
DECODE_KEYS sample_keys[7] =  
{  
    {BLANK},  
    {OFFON, {"Label 1"}, {1}},  
    {BLANK},  
    {OFFON, {"Label 2"}, {0}}  
};
```

OFF/ON Type Softkey

This softkey uses the first label and all other labels are ignored. It is full-bright when the value is 1; half-bright for value 0. Pressing the softkey toggles the value between 1 and 0.

Note

The program label for an ON/OFF softkey is *ONOFF*.

The softkey syntax uses the backwards single quote (‘) to indicate the start of the bottom line of a softkey label.

The following softkey structure would do the following.

- Label the first softkey as **Event Number** in full-bright video.
- Label the second softkey as **Lead Status** in half-bright video.

```
DECODE_KEYS keys_a[7] =  
{  
    {OFFON, {"Event'Number"}, {1}},  
    {OFFON, {"Lead'Status"}, {0}}  
};
```

A decode routine for this softkey structure is as follows.

```
decode_a(event,event_num)  
EVENT *event: /* Pointer to the event array element to display.*/  
int event_num; /* The event number; range 1 through 3000. */  
{  
    if(keys_a[0].value) /* If "Event Number" is ON. */  
    {  
        dispf("\nEvent %4d ",event_num);  
    }  
    if(keys_a[1].value) /* If "Lead Changes" is ON. */  
    {  
        dispf("\nLead Status: %4d ",event->lead_status);  
    }  
}
```

CYCLE Type Softkey

The softkey is always in full-bright video. Each time it is pressed it cycles to a different label until the NIL label. At this time it returns to the first label.

The value is always the current label; when the value is 1, the first label is displayed; when the value is 2, the second label, is displayed; and so on.

A maximum of seven non-NIL labels are supported. *The last label in any label string is always NIL.*

The following softkey structure would:

- Initialize softkey 3 label value = 2; **ASCII**.
- Pressing the softkey changes the value to 3; **EBCDIC**.
- Pressing the softkey changes the value to 1; **Hex**.
- Pressing the softkey changes the value to 2; **ASCII**, starting the cycle over again.

```
DECODE_KEYS keys_b[7] =
{
    {BLANK},
    {BLANK},
    {CYCLE, {"Hex","ASCII","EBCDIC"}, {2}}
};
```

The following decode routine simply displays the raw frame contents in the selected data code. The source (DTE or DCE) is not identified, there is no FCS reporting, and a row of asterisks is displayed below each frame.

```
decode_b(event, event_num)
EVENT *event;
int event_num;
{
    switch_keys_b[2].value)    /* Softkey 3 value. */
    {
        case 1:                /* Hex */
            dispf("\n%X*r ",event->length,event->frame_ptr);
            break;
        case 2:                /* ASCII */
            dispf("\n%*r ",event->length,event->frame_ptr);
            break;
        case 3:                /* EBCDIC */
            dispf("\n%E*r ",event->length,event->frame_ptr);
            break;
    }
    dispf("\n*****");
}
```

ZCYCLE Type Softkey

This softkey works like the CYCLE softkey except it has a distinct *OFF* setting. The off value is 0 and displays the softkey in half-bright video.

The first label listed is the OFF label; the next label is the value 1 label and so on.

The following softkey structure selects the data code and then defines how much raw frame data to display. The display can be no data, one line of data, or the entire frame no matter how many lines it takes.

Note The program label for a ZCYCLE softkey is *ZCYCL*.

- Softkey 4 is initialized to half-bright video (value = 0) with the label:

Raw Frame
Contents

- Pressing the key once changes the video to full-bright (value = 1) with the label:

One-Line
Undecoded

- Pressing the key again changes the video to full-bright (value = 2) with the label:

Full
Undecoded

- Pressing the key again returns to the half-bright display (value = 0) with the label:

Raw Frame
Contents

```
DECODE_KEYS keys_c[7] =
{
  {BLANK},
  {BLANK},
  {CYCLE, {"Hex","ASCII","EBCDIC"}, {2}}
  {ZCYCL, {"Raw Frame'Contents","One-Line'Undecoded", "Full'Undecoded"}, {0}}
};
```

LAYER Type Softkey

A *layer* softkey selects another set of softkeys. This second set or layer of softkeys is defined with the *same structure* that defined the first softkey set or layer.

The `DECODE_KEYS` structure address is in the *next layer* field, that is field *after* the label value field (see `gbops_top` structure, below). This field is only recognized when the softkey type is layer. All other softkey types ignore this field.

There is only one label for a layer softkey, the first label. No other label is recognized. The value, and therefore the video display, depends on the values of the non-CYCLE (i.e., BLANK, OFFON, ZCYCLE) softkeys on the lower layer, its *children*. For a full-bright video display (value = 1) at *least one* non-CYCLE must have a non-zero value. When *all* non-CYCLE softkey values are 0, the layer softkey value is 0 and the display is half-bright.

There is no specific limit on the number of layers that you can nest. The limiting factor is stack space because the softkey handler is recursive. However, realistically two layers is usually sufficient for most applications.

The following three structures define two softkey layers. The first, `gbops_top`, is part of the `USER.csource` file. Softkeys 4 and 5 call the second layer softkey structures `gbopsl2_keys` and `gbopsl3_keys`, respectively.

The functions of the `gbopsl2_keys` softkeys are as follows.

- Softkey 1: Select the data code.
- Softkey 2: Select the amount of raw Level 2 data to display.
- Softkey 3: No function.
- Softkey 4: Select the decode type.
- Softkey 5: No function
- Softkey 6: Select extended control ON/OFF.
- Softkey 7: Select multilink ON/OFF.

The `gbopsl3_keys` structure defines softkeys 1 and 2.

- Softkey 1: Select the data code.
- Softkey 2: Select the amount of raw Level 3 data to display.m

In each layer softkey 8 is labeled **Exit**. When on the lower softkey layer, pressing it returns you to the gbops_top layer. Pressing **Exit** when in gbops_top returns you to the top layer.

```
DECODE_KEYS gbops12_keys[7] =
{
    {CYCLE, {"Hex","ASCII","EBCDIC","Binary"}, {2} },
    {ZCYCL, {"Raw Bytes","One-Line'Raw Bytes",
             "All'Raw Bytes"}, {0}}
    {BLANK},
    {ZCYCL, {"Layer 2'Decode","LAPB'Decode","HDLC'Decode",
             "SDLC'Decode"}, {1}},
    {BLANK},
    {OFFON, {"Extended'Control"}, {0} },
    {OFFON, {"MultiLink'Info"}, {0} }
};

DECODE_KEYS gbops13_keys[7] =
{
    {CYCLE, {"Hex","ASCII","EBCDIC","Binary"}, {2} },
    {ZCYCL, {"Raw Bytes","One-Line'Raw Bytes","All'Raw Bytes"}, {2}}
};

DECODE_KEYS gbops_top[7] =
{
    {OFFON, {"Event/'Timestamp"}, {0} },
    {ZCYCL, {"Lead'Status","RS232/'MIL188C","RS449","V-35",
             "Lead #"}, {0} },
    {BLANK},
    {LAYER, {"GBOPS'Layer 2"}, {1}, gbops12_keys},
    {LAYER, {"GBOPS'Layer 3"}, {1}, gbops13_keys}
};
```

Modifying Softkeys

To modify the softkeys, edit the GBOPS_KEYS.csource file. You can add new labels, additional layers, or use different default values by the following the rules and procedures discussed above.

Running BOPS DECODE

The final phase is running the program and requires you to do the following.

1. Define the linkages.
2. Create the program file.
3. Start the program.

The first two items use the *BOPS.linkcommand* and *GBOPS.text* files and are only performed the first time you use BOPS DECODE or after you modify any of the source files. A description of these procedures follows.

The program is started using DataCommC C Manager function as described in the *DataCommC User's Guide*.

Defining Linkages

The BOPS.linkcommand file (see figure 3-3) is on the Solutions disc. This defines the object files to be linked. See the chapter 7, "The Linker" for a complete description of linkcommand files.

```
c:C/  
HP/DSDISP.object  
c:C/HP/DSEXAM.object  
c:C/HP/DSMAIN.object  
c:C/HP/DSRTD.object  
c:C/HP/USER.object  
c:C/HP/GBOPS_KEYS.object  
c:C/HP/GBOPS_DEC.object  
c:C/HP/MAIN_DEC.object  
c:C/HP/GBOPS.object  
c:C/HP/GBOPS_LEAD.object
```

Figure 3-2. BOPS.linkcommand File

Creating the Program File

DataCommC uses the GBOPS.text *MAKE* file (see figure 3-4) to compile the source files, link the object files, and create the *BOPS.program* file.

A *new* object file is created from the source code under the following conditions.

- No object file exists.
- The source code has been changed since the object file was created.

Once all object files are updated and the linkages made, a *new BOPS.program* file is created.

See the DataCommC User's Guide for a complete description of *MAKE* files.

```
#
# make file for GENERAL BOPS DECODE
#
OBJ=DSDISP.OBJECT DSEXAM.OBJECT DSMAIN.OBJECT DSRTD.OBJECT \
    USER.OBJECT \
    GBOPS_KEYS.OBJECT GBOPS_DEC.OBJECT MAIN_DEC.OBJECT GBOPS.OBJECT\
    GBOPS_LEAD.OBJECT
PROGRAM=BOPS

$(PROGRAM).program: $(OBJ)
    link $(PROGRAM).linkcommand
```

Figure 3-3. GBOPS.text File

Source and Include File Listings

The following are Solutions disc source and include file listings. These listings have been reformatted for easier reading and the comments expanded to better describe function and operation. *The code has not been changed.* The code on the Solutions disc has been tested for proper operation prior to shipment.

As with all software products, the programs may be revised prior to shipment. Therefore, it is possible that there may be *minor* differences.

Program Listing - MAIN_DEC.csource

```

/*****
*
*                               HP 18320A DataCommC
*                               Decoding Displays
*
*                               File: MAIN_DEC.csource
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
* The main_dec routine decodes and displays one event at a time. The User.csource
* file user( ) routine identifies this routine to the display.kernel.
*****/

#include "DECODE.include"
#include "message.include"
#include "GLOBUS.include"
#include "GBOPS.include"

#define TRUE 1
#define FALSE 0

extern decode_frame();
extern display_frame();
extern dispf();
extern DECODE_KEYS gbops_top[];    /* User keys */

/*****
* The "main_dec" routine
*****/

main_dec(event,event_num)
FRAME_EVENT *event;
unsigned int event_num;

{
    DEC_GBOPS out_ptr;
    int col;

    if (event->subtype == LEAD_CHANGE)
    {
        if (gbops_top[1].value)    /* LEAD STATUS is ON. */
        {
            if (event->length)    /* Not the first lead status. */
            {
                lead_change_disp(event,event_num,&gbops_top[0]);
            }
            else
            {
                if (gbops_top[0].value)    /* EVENT/TIMESTAMP is ON */

```

Program Listing - MAIN_DEC.csource (Continued)

```
        {
            dispf("\n%p%Us%-4ld%Us%02d:%02d:%02d.%03d",1,"event ",(long)event_num,
                "                ",event->hour,event->minute,event->second,
                event->msec);
        }
        lead_stat_disp(event->lead_status,1,&gbops_top[0]);
    }
}
else
{
    decode_frame(event,&out_ptr,((gbops_top[3].next_layer)[5].value &&
        ( (gbops_top[3].next_layer)[2].value != LAP) ),
        (gbops_top[3].next_layer)[2].value,
        (gbops_top[3].next_layer)[6].value );
    if (event->subtype == DTE_FRAME)
    {
        col = 1;    /* DTE */
    }
    else
    {
        col = 40;    /* DCE */
    }
    if (gbops_top[0].value)    /* EVENT/TIMESTAMP is ON */
    {
        dispf("\n%p%Us%-4ld%Us%02d:%02d:%02d.%03d",col,"event ",(long)event_num,
            "                ",event->hour,event->minute,event->second,
            event->msec);
    }

    if (gbops_top[1].value)    /* LEAD_STATUS is ON */
    {
        lead_stat_disp(event->lead_status,col,&gbops_top[0]);
    }
}
```

Program Listing - MAIN_DEC.csource (Continued)

```

/*****
* This section toggles the COMMAND/RESPONSE frame mode based on the subscriber
* physical connection.
*****/

    if ( ((gbops_top[3].next_layer)[4].value == 2) && /* Physical connection = DCE.
*/
        (out_ptr.is_command != UNDEF) )
    {
        if (out_ptr.is_command) /* Convert a command to a response. */
        {
            out_ptr.is_command = FALSE;
        }
        else /* Convert a response to a command. */
        {
            out_ptr.is_command = TRUE;
        }
    }

    /* Display decoded frame. */
    display_frame(gbops_top[3].next_layer,&out_ptr,col);

    /* Display undecoded Level 3 data. */
    if ( (gbops_top[4].value != NONE) && (out_ptr.result != INVALID) )
    {
        dispf ("\n%p%s", col, "L3 :");
        rawdisp (gbops_top[4].next_layer, &out_ptr.L3raw_lgth, (col+MARGIN),
            (LINELGTH-MARGIN));
    }
}
}

```

Program Listing - GBOPS.csource

```
/******
*
*          HP 18320A DataCommC
*          Decoding Displays
*
*          File:  GBOPS.csource
*
*          Copyright 1989
*          Hewlett-Packard Company
*
*          This file contains the frame and protocol determination routines.
*
*****/

#include "dlib.include"
#include "message.include"
#include "GLOBUS.include"
#include "GBOPS.include"

/* Global constants definitions. */
#define NULL          0
#define TRUE          1
#define FALSE         0
#define GBOPS_MINI_LGTH  2
#define BIT1           0x01
#define BITS21          0x03
#define BITS31          0x07
#define BITS41          0x0F
#define BITS71          0x7F
#define BITS81          0xFF

/* Address decoding field constants definitions. */
#define A_ADDR         0x03
#define B_ADDR         0x01
#define C_ADDR         0x0F
#define D_ADDR         0x07

/******
*
*   The "decode_frame" routine decodes Level 2 analyzer events using the protocol
*   definition passed in as a parameter. This routine neither checks nor issues an
*   error message if it cannot reach the Level 2 event. It is the responsibility of
*
*   the caller to ensure that the event can be reached.
*
*
*   INPUTS:
*   event_ptr:  Pointer to the event.
*   is_mod128:  Modulus definition; 0 = modulus 8, 1 = modulus 128.
*   protocol :  Protocol to decode: HDLC, LAP, LAPB, SDLC
*
*   OUTPUTS:
*   out_ptr  :  A pointer to a DEC_GBOPS structure defined in GBOPS.include
*
*****/
```

Program Listing - GBOPS.csource (Continued)

```

decode_frame(event_ptr, out_ptr, is_mod128, protocol, multi_is_on)

register FRAME_EVENT *event_ptr;
register DEC_GBOPS *out_ptr;
BOOL is_mod128;
int protocol;                /* LAP,LAPB,HDLC,SDLC */
BOOL multi_is_on;

{
    register BYTE *L2_ptr;    /* Pointer to a frame in the analyzer. */
    register int  L2_lgth;    /* Length of the analyzer frame. */
    unsigned char s_mask;     /* S-frame decoding mask. */

    /* Initialize output structure. */
    out_ptr->result = OK;
    out_ptr->is_multi = FALSE;
    out_ptr->is_command = UNDEF;

    /* Initialize Level 2 Raw Data */
    out_ptr->L2raw_lgth = event_ptr->length;
    out_ptr->L2raw_ptr = event_ptr->frame;

    /* Initialize Current Frame Pointer */
    L2_ptr = event_ptr->frame;
    L2_lgth = event_ptr->length;
    if (event_ptr->subtype == DCE_FRAME || event_ptr->subtype == SDCE_FRAME)
    {
        out_ptr->is_dceframe = TRUE;    /* DCE Frame */
    }
    else
    {
        out_ptr->is_dceframe = FALSE;    /* DTE Frame */
    }

    out_ptr->fcs = event_ptr->fcs;    /* Check FCS */

    if (out_ptr->fcs != GOOD_FCS)
    {
        out_ptr->result = INVALID;
        return ;
    }

    if (L2_lgth < GBOPS_MINI_LGTH)
    {
        out_ptr->result = INVALID;
        return ;
    }

    /* Decode address field. */
    out_ptr->addr = *L2_ptr;

    if (protocol == LAPB)    /* Check LAPB command, response, multilink, and address. */

```

Program Listing - GBOPS.csource (Continued)

```
{
    if (((*L2_ptr != A_ADDR) && (*L2_ptr != B_ADDR)) &&
        ((*L2_ptr != C_ADDR) && (*L2_ptr != D_ADDR)))
    {
        out_ptr->result |= ADDR_ERR;      /* Invalid LAPB address. */
    }
    if ( (*L2_ptr > A_ADDR) && !(out_ptr->result | ADDR_ERR))
    {
        out_ptr->is_multi = TRUE;      /* Multilink case */
        if (((*L2_ptr == C_ADDR) &&
            (event_ptr->subtype == DCE_FRAME || event_ptr->subtype == SDCE_FRAME))
            || ((*L2_ptr == D_ADDR) &&
            (event_ptr->subtype == DTE_FRAME || event_ptr->subtype == SDTE_FRAME)))
        {
            out_ptr->is_command = TRUE;
        }
        else
        {
            out_ptr->is_command = FALSE;
        }
    }
    else
    {
        out_ptr->is_multi = FALSE;      /* Single Link case */
        if (((*L2_ptr == A_ADDR) &&
            (event_ptr->subtype == DCE_FRAME || event_ptr->subtype == SDCE_FRAME))
            || ((*L2_ptr == B_ADDR) &&
            (event_ptr->subtype == DTE_FRAME || event_ptr->subtype == SDTE_FRAME)))
        {
            out_ptr->is_command = TRUE;
        }
        else
        {
            out_ptr->is_command = FALSE;
        }
    }
}
/* End of LAPB frame. */
else
{
    if (protocol == LAP)      /* Check LAP frame command, response, and address. */
    {
        if ( (*L2_ptr != A_ADDR) && (*L2_ptr != B_ADDR) )
        {
            out_ptr->result |= ADDR_ERR;      /* Invalid LAP address. */
        }
        out_ptr->is_multi = FALSE;
        if (((*L2_ptr == A_ADDR) &&
            (event_ptr->subtype == DCE_FRAME || event_ptr->subtype == SDCE_FRAME))
            || ((*L2_ptr == B_ADDR) &&
            (event_ptr->subtype == DTE_FRAME || event_ptr->subtype == SDTE_FRAME)))
        {
            out_ptr->is_command = TRUE;
        }
        else
    }
```

Program Listing - GBOPS.csource (Continued)

```

    {
        out_ptr->is_command = FALSE;
    }
}

/* End of LAP frame. */

L2_ptr++;
L2_lgth--;
switch (*L2_ptr & BITS21) /* Control field decoding switch. */
{
    case 0x01: /* Supervisor format; p/f bit and N(r). */
        out_ptr->ctrl_format = SUPERV;

        if (is_mod128)
        {
            s_mask = BITS81;
        }
        else
        {
            s_mask = BITS41;
        }

        switch (*L2_ptr & s_mask) /* Start switch. */
        {
            case 1:
                out_ptr->ctrl_id = RR;
                break;
            case 5:
                out_ptr->ctrl_id = RNR;
                break;
            case 9:
                out_ptr->ctrl_id = REJ;
                break;
            case 13:
                if (protocol == HDLC)
                {
                    out_ptr->ctrl_id = SREJ;
                }
                else
                {
                    out_ptr->ctrl_id = UNDEF;
                }
                break;
            default:
                out_ptr->ctrl_id = UNDEF;
                break;
        } /* End switch */
}
if (is_mod128) /* Supervisory Modulus 128 format. */
{
    if (L2_lgth < 2) /* 2 byte control field. */
    {
        out_ptr->result = INVALID;
        return;
    }
}

```

Program Listing - GBOPS.csource (Continued)

```
        L2_ptr++;
        L2_lgth--;
        out_ptr->pf_bit = *L2_ptr & BIT1;
        out_ptr->Nr = (*L2_ptr >> 1) & BITS71;
    }
    else
    {
        /* Supervisor Modulus 8 format. */
        out_ptr->pf_bit = (*L2_ptr >> 4) & BIT1;
        out_ptr->Nr = (*L2_ptr >> 5) & BITS31;
    }
    L2_ptr++;
    L2_lgth--;
    break;

/* Unnumbered format: 1 byte (id), 2 byte for SDLC and HDLC mod-128. */
case 0x03:
    out_ptr->ctrl_format = UNUMB;
    if ( (protocol == HDLC) && is_mod128 )
    {
        out_ptr->pf_bit = BIT1 & *(L2_ptr + 1) ;
    }
    else
    {
        out_ptr->pf_bit = BIT1 & (*L2_ptr >> 4);
    }
    switch(protocol)      /* Start protocol switch. */
    {
        case HDLC:
            out_ptr->ctrl_id = HDLC_U(L2_ptr);
            break;
        case SDLC:
            out_ptr->ctrl_id = SDLC_U(L2_ptr);
            break;
        case LAPB:
            out_ptr->ctrl_id = LAPB_U(L2_ptr);
            break;
        case LAP:
            out_ptr->ctrl_id = LAP_U(L2_ptr);
            break;
        default:
            out_ptr->ctrl_id = UNDEF;
            break;
    }
    /* End of protocol switch. */

    if ( (protocol == HDLC) && is_mod128 )
    {
        L2_ptr++;
        L2_lgth--;
    }
    L2_ptr++;
    L2_lgth--;
    break;
```

Program Listing - GBOPS.csource (Continued)

```

default:                /* Information format: N(r) + N(s). */
    out_ptr->ctrl_format = INFO;
    out_ptr->ctrl_id = I;

    if (is_mod128)
    {
        if (L2_lgth < 2)    /* Information modulus 128 format. */
        {
            out_ptr->result = INVALID;
            return;
        }
        out_ptr->Ns = (*L2_ptr >> 1) & BITS71;
        L2_ptr++;
        L2_lgth--;
        out_ptr->pf_bit = *L2_ptr & BIT1;
        out_ptr->Nr = (*L2_ptr >> 1) & BITS71;
    }
    else
    {
        /* Information modulus 8 format. */
        out_ptr->Ns = (*L2_ptr >> 1) & BITS31;
        out_ptr->pf_bit = (*L2_ptr >> 4) & BIT1;
        out_ptr->Nr = (*L2_ptr >> 5) & BITS31;
    }
    L2_ptr++;
    L2_lgth--;
}
/* End control field decoding. */
if (out_ptr->ctrl_id == I)    /* Information field decoding. */
{
    if ((out_ptr->is_multi) && (multi_is_on))
    {
        if (L2_lgth < 2)    /* Information frame multilink Control Field. */
        {
            out_ptr->result |= MULTI_ERR;    /* Multilink Control Field error. */
        }
        else
        {
            out_ptr->V_bit = (*L2_ptr >> 4) & BIT1;
            out_ptr->S_bit = (*L2_ptr >> 5) & BIT1;
            out_ptr->R_bit = (*L2_ptr >> 6) & BIT1;
            out_ptr->C_bit = (*L2_ptr >> 7) & BIT1;
            out_ptr->MNs = (*L2_ptr & BITS41) << 8;
            out_ptr->MNs |= (*L2_ptr + 1);
            L2_ptr += 2;
            L2_lgth -= 2;
        }
    }
    /* End Multilink. */
}
/* End Information field decoding. */

if (out_ptr->ctrl_id == FRMR_CMDR)    /* Start FRMR Information */
{
    if (is_mod128)    /* Check for Modulus 128 */
    {
        if (L2_lgth < 5)    /* Check for five bytes */

```

Program Listing - GBOPS.csource (Continued)

```
{
    out_ptr->result |= FRMR_ERR; /* FRMR Information error */
}
else
{
    out_ptr->frmr.ctrl = *L2_ptr << 8;
    out_ptr->frmr.ctrl |= *(L2_ptr + 1);
    out_ptr->frmr.Vs = (*(L2_ptr + 2) >> 1) & BITS71;
    out_ptr->frmr.Vr = (*(L2_ptr + 3) >> 1) & BITS71;
    if (protocol == LAPB)
    {
        out_ptr->frmr.is_command = !(*(L2_ptr + 3) & BIT1);
    }
    else
    {
        out_ptr->frmr.is_command = UNDEF;
    }
    L2_ptr += 4;
    L2_lgth -= 4;
    out_ptr->frmr.is_ctrl_undef = *L2_ptr & BIT1;
    out_ptr->frmr.is_ctrl_incor = (*L2_ptr >> 1) & BIT1;
    out_ptr->frmr.is_infoGtmax = (*L2_ptr >> 2) & BIT1;
    out_ptr->frmr.is_Nr_invalid = (*L2_ptr >> 3) & BIT1;
    L2_ptr ++;
    L2_lgth --;
}
}
else /* FRMR Information Modulus 8 format */
{
    if (L2_lgth < 3) /* Check for three bytes */
    {
        out_ptr->result |= FRMR_ERR; /* FRMR Information error */
    }
    else
    {
        out_ptr->frmr.ctrl = *L2_ptr;
        out_ptr->frmr.Vs = (*(L2_ptr + 1) >> 1) & BITS31;
        out_ptr->frmr.Vr = (*(L2_ptr + 1) >> 5) & BITS31;
        if (protocol == LAPB)
        {
            out_ptr->frmr.is_command = !((*(L2_ptr + 1) >> 4) & BIT1);
        }
        else
        {
            out_ptr->frmr.is_command = UNDEF;
        }
        L2_ptr += 2;
        L2_lgth -= 2;
        out_ptr->frmr.is_ctrl_undef = *L2_ptr & BIT1;
        out_ptr->frmr.is_ctrl_incor = (*L2_ptr >> 1) & BIT1;
        out_ptr->frmr.is_infoGtmax = (*L2_ptr >> 2) & BIT1;
        out_ptr->frmr.is_Nr_invalid = (*L2_ptr >> 3) & BIT1;
        L2_ptr ++;
        L2_lgth --;
    }
}
```

Program Listing - GBOPS.csource (Continued)

```

    }
}
/* End FRMR Information */
if (L2_lgth > 0) /* Rest of information field - Level 3 l l */
{
    if ( !( (out_ptr->ctrl_format == INFO) ||
            (out_ptr->ctrl_id == FRMR_CMDR) ||
            (out_ptr->ctrl_id == CFGR) ||
            (out_ptr->ctrl_id == XID) ||
            (out_ptr->ctrl_id == UI) ||
            (out_ptr->ctrl_id == TEST) ) ) /* if not one of valid frames for info */
    {
        out_ptr->result |= UNEX_INFO;
    }
    out_ptr->L3raw_lgth = L2_lgth;
    out_ptr->L3raw_ptr = L2_ptr;
}
else
{
    out_ptr->L3raw_lgth = 0;
    out_ptr->L3raw_ptr = NULL;
}
} /* GBOPS_LEX */

```

Program Listing - GBOPS.csource (Continued)

```
/******  
* The "LAPB_U" routine determines the LAPB U-frame type. *  
*****/  
  
LAPB_U(L2_ptr)  
register BYTE *L2_ptr;  
  
{  
    switch (*L2_ptr)  
    {  
        case 0x0F:  
        case 0x1F:  
            return(DM_SARM);  
            break;  
        case 0x2F:  
        case 0x3F:  
            return(SABM);  
            break;  
        case 0x6F:  
        case 0x7F:  
            return(SABME);  
            break;  
        case 0x43:  
        case 0x53:  
            return(DISC_RD);  
            break;  
        case 0x63:  
        case 0x73:  
            return(UA);  
            break;  
        case 0x87:  
        case 0x97:  
            return(FRMR_CMDR);  
            break;  
        default:  
            return(UNDEF);  
            break;  
    }  
} /* End switch */
```

Program Listing - GBOPS.csource (Continued)

```

/*****
* The "LAP_U" routine determines the LAP U-frame type.
*****/

LAP_U(L2_ptr)
register BYTE *L2_ptr;

{
    switch (*L2_ptr)
    {
        case 0x0F:
        case 0x1F:
            return(DM_SARM);
            break;
        case 0x43:
        case 0x53:
            return(DISC_RD);
            break;
        case 0x63:
        case 0x73:
            return(UA);
            break;
        case 0x87:
        case 0x97:
            return(FRMR_CMDR);
            break;
        default:
            return(UNDEF);
            break;
    }
}

/* End switch. */
```

Program Listing - GBOPS.csource (Continued)

```
/*
*****
* The "HDL_C_U" routine determines the HDLC_U-frame type.
*****
*/

HDL_C_U(L2_ptr)
register BYTE *L2_ptr;

{
    switch (*L2_ptr)
    {
        case 0x0F:
        case 0x1F:
            return(DM_SARM);
            break;
        case 0x2F:
        case 0x3F:
            return(SABM);
            break;
        case 0x6F:
        case 0x7F:
            return(SABME);
            break;
        case 0x43:
        case 0x53:
            return(DISC_RD);
            break;
        case 0x63:
        case 0x73:
            return(UA);
            break;
        case 0x87:
        case 0x97:
            return(FRMR_CMDR);
            break;
        case 0xAF:
        case 0xBF:
            return(XID);
            break;
        case 0x03:
        case 0x13:
            return(UI);
            break;
        case 0x07:
        case 0x17:
            return(SIM_RIM);
            break;
        case 0x4F:
        case 0x5F:
            return(SARME);
            break;
        case 0xCF:
        case 0xDF:
            return(SNRME);
            break;
    }
}
```

Program Listing - GBOPS.csource (Continued)

```
case 0x83:
case 0x93:
    return(SNRM);
    break;
case 0x23:
case 0x33:
    return(UP);
    break;
case 0x8F:
case 0x9F:
    return(RSET);
    break;
default:
    return(UNDEF);
    break;
}
/* End switch. */
}
```

Program Listing - GBOPS.csource (Continued)

```

/*****
* The "SDLC_U" routine determines the SDLC U-frame type.
*****/

SDLC_U(L2_ptr)
register BYTE *L2_ptr;

{
    switch (*L2_ptr)                /* Start switch. */
    {
        case 0x0F:
        case 0x1F:
            return(DM_SARM);
            break;
        case 0x43:
        case 0x53:
            return(DISC_RD);
            break;
        case 0x63:
        case 0x73:
            return(UA);
            break;
        case 0x87:
        case 0x97:
            return(FRMR_CMDR);
            break;
        case 0xAF:
        case 0xBF:
            return(XID);
            break;
        case 0xC7:
        case 0xD7:
            return(CFGR);
            break;
        case 0x03:
        case 0x13:
            return(UI);
            break;
        case 0xE3:
        case 0xF3:
            return(TEST);
            break;
        case 0x07:
        case 0x17:
            return(SIM_RIM);
            break;
        case 0x83:
        case 0x93:
            return(SNRM);
            break;
        case 0x23:
        case 0x33:
            return(UP);
            break;
    }
}
```

Program Listing - GBOPS.csource (Continued)

```

    case 0xEF:
    case 0xFF:
        return(BCN);
        break;
    default:
        return(UNDEF);
        break;
}
/* End switch. */

```

Program Listing - GBOPS_DEC.csource

```

*****
*                               HP 18320A DataCommC                               *
*                               Decoding Displays                               *
*                               *                               *
*                               File: GBOPS_DEC.csource                       *
*                               *                               *
*                               Copyright 1989                               *
*                               Hewlett-Packard Company                       *
*                               *                               *
*   This two major routines in this file display level 2 information and a string of *
*   raw Level 2 bytes using display_frame( ) and rawdisp( ), respectively.      *
*****/

#include "dlib.include"
#include "DECODE.include"
#include "GLOBUS.include"
#include "GBOPS.include"
#include "video.include"
#include "conio.include"

/* Global constants definitions. */
#define NULL      0
#define TRUE      1
#define FALSE     0

/*****
* The "display_frame" routine displays level analyzer event information per the
* gbops_lex output structure.
*
* GLOBALS:
*   LINELGTH   : The maximum displayed line length.
* INPUTS:
*   softkeys   : An array of gbops softkeys.
*
*   outlex_ptr : Pointer to a GBOPS.include defined DEC_GBOPS structure.
*
*   firstcol   : The column where the screen line begins.
* OUTPUTS:
*   Lines in screen memory through dispf( ).
*****/

```

Program Listing - GBOPS_DEC.csource (Continued)

```
display_frame(softkeys, outlex_ptr, firstcol)

register DECODE_KEYS softkeys[];
register DEC_GBOPS *outlex_ptr;
register int firstcol;

    if (softkeys[3].value)      /* Display level 2 decode label in margin. */
    {
        switch(softkeys[2].value)
        {
            case 1:
                dispf ("\n%p%s", firstcol, "LAPB:");
                break;
            case 2:
                dispf ("\n%p%s", firstcol, "LAP :");
                break;
            case 3:
                dispf ("\n%p%s", firstcol, "HDLC:");
                break;
            case 4:
                dispf ("\n%p%s", firstcol, "SDLC:");
                break;
            default:
                dispf ("\n%p%s", firstcol, "ERR :");
                break;
        }
    }
    else
    {
        if (softkeys[1].value)    /* Display selected raw display. */
        {
            dispf ("\n%p%s", firstcol, "L2 :");
        }
    }
    switch (outlex_ptr->fcs)      /* Start FCS switch. */
    {
        case BAD_FCS:
            dispf ("%Bs%Bs", "(", "BAD FCS", ")");    /* Display BAD_FCS. */
            break;
        case ABORT_FCS:
            dispf ("%Bs%Bs", "(", "ABORT FCS", ")");    /* Display ABORT_FCS. */
            break;
        default:
            break;
    }
    /* End FCS switch. */
    if (softkeys[1].value != NONE) /* Display UNDECODED Level 2 data. */
    {
        if (outlex_ptr->fcs != GOOD_FCS)
        {
            dispf("\n");
        }
        rawdisp (&softkeys[0], &outlex_ptr->L2raw_lgth, (firstcol+MARGIN),
            (LINELGTH-MARGIN));
    }
}
```

Program Listing - GBOPS_DEC.csource (Continued)

```
if (!softkeys[3].value)      /* Update keys or display Level 2 raw data. */
{
    return(0);
}

if (softkeys[2].value)      /* Display DECODED Level 2 data. */
{
    if (outlex_ptr->result == INVALID)      /* NO DECODING. */
    {
        switch (outlex_ptr->fcs)
        {
            case BAD_FCS:                  /* CAUSE: Bad FCS. */
            case ABORT_FCS:                /* CAUSE: Abort FCS. */
                break;

            default:
                dispf ("\n%pBs", (firstcol+WARN_MARG), "!");
                /* CAUSE: MAJOR frame error. */
                dispf ("%Hs", "can't be decoded: LENGTH ERROR");
        }
    }
    else
    {
        /* DECODING DONE. Possible MINOR frame error. */
        if (softkeys[1].value != NONE)
        {
            /* Open a new line for previously displayed bytes. */
            dispf ("\n%p", (firstcol+MARGIN));
        }
        switch (outlex_ptr->ctrl_id)      /* Control identification switch. */
        {
            case I:
                dispf (" INFO  ");
                break;
            case RR:
                dispf (" RR    ");
                break;
            case RNR:
                dispf (" RNR   ");
                break;
            case REJ:
                dispf (" REJ   ");
                break;
            case SREJ:
                dispf (" SREJ  ");
                break;
            case SABME:
                dispf (" SABME ");
                break;
            case SABM:
                dispf (" SABM  ");
                break;
        }
    }
}
```

Program Listing - GBOPS_DEC.csource (Continued)

```
case DM_SARM:
    switch(softkeys[2].value)
    {
        case 1: /* LAPB */
        case 4: /* SDLC */
            dispf (" DM      ");
            break;
        case 2: /* LAP */
            dispf (" SARM   ");
            break;
        case 3: /* HDLC */
            dispf (" DM/SARM");
            break;
        default:
            dispf (" ERR    ");
            break;
    }
    break;
case UI:
    dispf (" UI      ");
    break;
case DISC_RD:
    switch(softkeys[2].value)
    {
        case 1: /* LAPB */
        case 2: /* LAP */
            dispf (" DISC  ");
            break;
        case 3: /* HDLC */
        case 4: /* SDLC */
            dispf (" DISC/RD");
            break;
        default:
            dispf (" ERR    ");
            break;
    }
    break;
case UA:
    dispf (" UA      ");
    break;
case FRMR_CMDR:
    if (softkeys[2].value == LAP)
    {
        dispf (" CMDR   ");
    }
    else
    {
        dispf (" FRMR   ");
    }
    break;
case CFGR:
    dispf (" CFGR   ");
    break;
```

Program Listing - GBOPS_DEC.csource (Continued)

```

case TEST:
    dispf (" TEST  ");
    break;
case SIM_RIM:
    dispf (" SIM/RIM");
    break;
case SARME:
    dispf (" SARME  ");
    break;
case SNRME:
    dispf (" SNRME  ");
    break;
case SNRM:
    dispf (" SNRM   ");
    break;
case UP:
    dispf (" UP     ");
    break;
case RSET:
    dispf (" RSET   ");
    break;
case BCN:
    dispf (" BCN    ");
    break;
case XID:
    dispf (" XID    ");
    break;
default:
    dispf (" UNDEF  ");
}
if (outlex_ptr->is_command == TRUE)      /* Command - character "C." */
{
    dispf (" %s", "C");
}
else
{
    if (outlex_ptr->is_command == FALSE)  /* Response - character "R." */
    {
        dispf (" %s", "R");
    }
}
/* Command or Response frame address. */
dispf (" %Hs%02x", "Addr", outlex_ptr->addr);
if (outlex_ptr->pf_bit)      /* P/F bit */
{
    if ( (softkeys[2].value == LAP) || (softkeys[2].value == LAPB) )
    {
        if (outlex_ptr->is_command)
        {
            dispf (" P");      /* Command frame P bit. */
        }
        else

```

Program Listing - GBOPS_DEC.csource (Continued)

```

        {
            dispf (" F");          /* Response frame F bit. */
        }
    }
    else /* HDLC, SDLC */
    {
        dispf (" P/F");
    }
}
switch (outlex_ptr->ctrl_format) /* Control format switch. */
{
    case INFO: /* N(r) and (s) definition. */
        dispf (" %Hs%d %Hs%d", "Nr", outlex_ptr->Nr, "Ns", outlex_ptr->Ns);
        break;
    case SUPERV: /* N(r) definition. */
        dispf (" %Hs%d", "Nr", outlex_ptr->Nr);
        break;
    default: /* UNNUMBERED format; neither N(r) nor N(s) are defined. */
        ;
} /* End control format switch. */

switch (outlex_ptr->ctrl_id) /* Level 2 information switch. */
{
    case I:
        /* Multilink decode ON. */
        if ( (softkeys[6].value) && (outlex_ptr->is_multi) )
        {
            if (outlex_ptr->result & MULTI_ERR)
            {
                /* No decoding; MAJOR multilink error. */
                dispf ("\n%pBs%Hs", (firstcol+WARN_MARG), "!",
                    "MULTILINK info undecoded: LENGTH ERROR");
                dispf ("\n"); /* Open line required for rawdisp. */
                rawdisp (&softkeys[0], &outlex_ptr->L3raw_lgth,
                    (firstcol+INDENT1), (LINELGTH-INDENT1));
            }
            else
            {
                dispf ("\n%pBs%Hs ",(firstcol + MARGIN),"MNs", outlex_ptr->MNs);
                if (outlex_ptr->V_bit)
                {
                    dispf ("\n%pBs %Hs", (firstcol+INDENT1), "V",
                        "void sequencing bit");
                }
                if (outlex_ptr->S_bit)
                {
                    dispf ("\n%pBs %Hs", (firstcol+INDENT1), "S",
                        "sequence check option bit");
                }
                if (outlex_ptr->R_bit)
                {
                    dispf ("\n%pBs %Hs", (firstcol+INDENT1), "R",
                        "MLP reset request bit");
                }
            }
        }
    }
}

```

Program Listing - GBOPS_DEC.csource (Continued)

```

        if (outlex_ptr->C_bit)
        {
            dispf ("\n%p%s %Hs", (firstcol+INDENT1), "C",
                "MLP reset confirmation bit");
        }
    }
}
break;
case FRMR_CMDR:
    if (outlex_ptr->result & FRMR_ERR)
    {
        /* No decoding; MAJOR FRMR error. */
        dispf ("\n%pBs%Hs", (firstcol+WARN_MARG), "!",
            "FRMR info undecoded: LENGTH ERROR");
        dispf ("\n"); /* Open line required for rawdisp. */
        rawdisp (&softkeys[0], &outlex_ptr->L3raw_lgth,
            (firstcol+INDENT1), (LINELGTH-INDENT1));
    }
    else
    {
        if ( (softkeys[5].value) && (softkeys[2].value != LAP) )
        {
            dispf ("\n%p%Hs%04lx %Hs%d %Hs%d", (firstcol + MARGIN),
                "control", outlex_ptr->frmr.ctrl, "Vs", outlex_ptr->frmr.Vs,
                "Vr", outlex_ptr->frmr.Vr);
        }
        else
        {
            dispf ("\n%p%Hs%02lx %Hs%d %Hs%d", (firstcol + MARGIN),
                "control", outlex_ptr->frmr.ctrl, "Vs", outlex_ptr->frmr.Vs,
                "Vr", outlex_ptr->frmr.Vr);
        }
        if (softkeys[2].value == 1 /* LAPB */)
        {
            if (outlex_ptr->frmr.is_command)
            {
                dispf (" C");
            }
            else
            {
                dispf (" R");
            }
        }
        if (outlex_ptr->frmr.is_ctrl_undef)
        {
            dispf ("\n%p%s %Hs", (firstcol+INDENT1), "W",
                "control field undefined");
        }
        if (outlex_ptr->frmr.is_ctrl_incor)
        {
            dispf ("\n%p%s %Hs", (firstcol+INDENT1), "X",
                "control field invalid");
        }
        if (outlex_ptr->frmr.is_infoGTmax)
    }

```

Program Listing - GBOPS_DEC.csource (Continued)

```
        {
            dispf ("\n%p%s %Hs", (firstcol+INDENT1), "Y",
                "info field too long");
        }
        if (outlex_ptr->frmr.is_Nr_invalid)
        {
            dispf ("\n%p%s %Hs", (firstcol+INDENT1), "Z",
                "invalid Nr");
        }
    }
    break;
default:
    ;
} /* End Level 2 Information switch. */
if (outlex_ptr->result & UNEX_INFO) /* Error Messages. */
{
    dispf ("\n%p%Bs%Hs", (firstcol+WARN_MARG), "!",
        "unexpected information");
}
if (outlex_ptr->result & ADDR_ERR)
{
    dispf ("\n%p%Bs%Hs", (firstcol+WARN_MARG), "!",
        "invalid address");
}
}
}
} /* GBOPS_DEC */
```

Program Listing - GBOPS_KEYS.csource

```

/*****
*
*                               HP 18320A DataCommC
*                               Decoding Displays
*
*                               File:  GBOPS_KEYS.csource
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
* This file defines the Display Decoding softkeys.
*****/

#include "DECODE.include"
#include "MACROS.include"

/*****
* The "gbopsL2_keys" structure defines the Level 2 softkeys.
*****/

DECODE_KEYS gbopsL2_keys[7] = {
    {CYCLE, {"Hex", "ASCII", "EBCDIC", "Binary"}, {2} },
    {ZCYCL, {"Raw Bytes", "One-Line Raw Bytes", "All Raw Bytes"}, {0}},
    {CYCLE, {"LAPB Decode", "LAP Decode", "HDLC Decode", "SDLC Decode"}, {1} },
    {OFFON, {"BOPS Decode"}, {1} },
    {CYCLE, {"Subscr is Phys DTE", "Subscr is Phys DCE"}, {1} },
    {OFFON, {"Extended Control"}, {0} },
    {OFFON, {"Multilink Info"}, {0} }
};

/*****
* The "gbopsL3_keys" structure defines the Level 3 softkeys.
*****/

DECODE_KEYS gbopsL3_keys[7] = {
    {CYCLE, {"Hex", "ASCII", "EBCDIC", "Binary"}, {2} },
    {ZCYCL, {"Raw Bytes", "One-Line Raw Bytes", "All Raw Bytes"}, {2}}
};

/*****
* The "gbops_top" structure defines the top level softkeys.
*****/

DECODE_KEYS gbops_top[7] = {
    {OFFON, {"Event/ Timestamp"}, {0} },
    {ZCYCL, {"Lead Status", "RS232/ MIL188C", "RS449", "V_35", "Lead #"}, {0} },
    {BLANK},
    {LAYER, {"BOPS Layer 2"}, {1}, gbopsL2_keys},
    {LAYER, {"BOPS Layer 3"}, {1}, gbopsL3_keys}
};

```

Program Listing - GBOPS_LEAD.csource

```

/*****
*
*          HP 18320A DataCommC
*          Decoding Displays
*
*          File:  GBOPS_LEAD.csource
*
*          Copyright 1989
*          Hewlett-Packard Company
*
*
* This file uses the routines lead_stat_disp( ) and lead_change_disp( ) routines to
* display the lead changes and the status of the leads.
*****/

#include "message.include"
#include "leads.include"
#include "retval.include"
#include "GLOBUS.include"
#include "DECODE.include"

/*****
* The "lead_change_disp" routine displays interface pod lead changes.
*****/

lead_change_disp(event,event_num,softkeys)
LEAD_EVENT *event;
int event_num;
DECODE_KEYS softkeys[];

{
    int col;

    if (event->lead & (RTS|DTR|SRS|RS|TR|IS|SF|RL|SS) )
    {
        col = 1;
    }
    else
    {
        col = 40;
    }

    if (softkeys[0].value)    /* EVENT/TIMESTAMP */
    {
        dispf("\n%p%Us%U-4d%Us%02d:%02d:%02d.%03d",col,"event ",event_num,
            "          ",event->hour,event->minute,event->second,event->msec);
    }
    dispf("\n%p%Hs",col,"LEAD CHANGE ");

    switch (softkeys[1].value)    /* Start Pod Identification Switch. */
    {
        case 1:    /* Start RS-232 or MIL-188C Interface Pod Switch. */

```

Program Listing - GBOPS_LEAD.csource (Continued)

```
switch (event->lead)
{
    case 0x0001l :
        dispf("%*p%s ",col+12,"RTS ");
        break;

    case 0x0002l :
        dispf("%*p%s ",col+12,"CTS ");
        break;

    case 0x0004l :
        dispf("%*p%s ",col+12,"DSR ");
        break;

    case 0x0008l :
        dispf("%*p%s ",col+12,"DTR ");
        break;

    case 0x0010l :
        dispf("%*p%s ",col+12,"RI ");
        break;

    case 0x0020l :
        dispf("%*p%s ",col+12,"CD ");
        break;

    case 0x0040l :
        dispf("%*p%s ",col+12,"SQ ");
        break;

    case 0x0080l :
        dispf("%*p%s ",col+12,"DRS ");
        break;

    case 0x0100l :
        dispf("%*p%s ",col+12,"SRS ");
        break;

    case 0x0200l :
        dispf("%*p%s ",col+12,"SCS ");
        break;

    case 0x0400l :
        dispf("%*p%s ",col+12,"SCD ");
        break;

    default :
        break;
}
break; /* End RS-232 or MIL-188C Interface Pod Switch. */
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
case 2 :      /* Start RS-449 Interface Pod Switch. */
```

```
switch (event->lead)
{
```

```
    case 0x0001l :
        dispf("%*p%s ",col+12,"RS ");
        break;
```

```
    case 0x0002l :
        dispf("%*p%s ",col+12,"CS ");
        break;
```

```
    case 0x0004l :
        dispf("%*p%s ",col+12,"DM ");
        break;
```

```
    case 0x0008l :
        dispf("%*p%s ",col+12,"TR ");
        break;
```

```
    case 0x0010l :
        dispf("%*p%s ",col+12,"IC ");
        break;
```

```
    case 0x0020l :
        dispf("%*p%s ",col+12,"RR ");
        break;
```

```
    case 0x0040l :
        dispf("%*p%s ",col+12,"SQ ");
        break;
```

```
    case 0x0080l :
        dispf("%*p%s ",col+12,"SI ");
        break;
```

```
    case 0x0100l :
        dispf("%*p%s ",col+12,"SRS ");
        break;
```

```
    case 0x0200l :
        dispf("%*p%s ",col+12,"SCS ");
        break;
```

```
    case 0x0400l :
        dispf("%*p%s ",col+12,"SRR ");
        break;
```

```
    case 0x0800l :
        dispf("%*p%s ",col+12,"IS ");
        break;
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
    case 0x1000l :
        dispf("%*p%s ",col+12,"SF ");
        break;

    case 0x2000l :
        dispf("%*p%s ",col+12,"RL ");
        break;

    case 0x4000l :
        dispf("%*p%s ",col+12,"SS ");
        break;

    default :
        break;
}
break;      /* End RS-449 Interface Pod Switch. */

case 3 :      /* Start V.35 Interface Pod Switch. */

    switch (event->lead)
    {
        case 0x0001l :
            dispf("%*p%s ",col+12,"RS ");
            break;

        case 0x0002l :
            dispf("%*p%s ",col+12,"CS ");
            break;

        case 0x0004l :
            dispf("%*p%s ",col+12,"DSR ");
            break;

        case 0x0008l :
            dispf("%*p%s ",col+12,"DTR ");
            break;

        case 0x0010l :
            dispf("%*p%s ",col+12,"RI ");
            break;

        case 0x0020l :
            dispf("%*p%s ",col+12,"CD ");
            break;

        case 0x0040l :
            dispf("%*p%s ",col+12,"LT ");
            break;

        default :
            break;
    }
}
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
break;      /* End V.35 Interface Pod Switch. */

default:    /* Start Lead Number Switch. */

switch (event->lead)
{
    case 0x0001l :
        dispf("%*p%s ",col+12,"1  ");
        break;

    case 0x0002l :
        dispf("%*p%s ",col+12,"2  ");
        break;

    case 0x0004l :
        dispf("%*p%s ",col+12,"3  ");
        break;

    case 0x0008l :
        dispf("%*p%s ",col+12,"4  ");
        break;

    case 0x0010l :
        dispf("%*p%s ",col+12,"5  ");
        break;

    case 0x0020l :
        dispf("%*p%s ",col+12,"6  ");
        break;

    case 0x0040l :
        dispf("%*p%s ",col+12,"7  ");
        break;

    case 0x0080l :
        dispf("%*p%s ",col+12,"8  ");
        break;

    case 0x0100l :
        dispf("%*p%s ",col+12,"9  ");
        break;

    case 0x0200l :
        dispf("%*p%s ",col+12,"10 ");
        break;

    case 0x0400l :
        dispf("%*p%s ",col+12,"11 ");
        break;

    case 0x0800l :
        dispf("%*p%s ",col+12,"12 ");
        break;
}
```

Program Listing - GBOPS_LEAD.csource (Continued)

```

        case 0x1000l :
            dispf("%*p%s ",col+12,"13 ");
            break;

        case 0x2000l :
            dispf("%*p%s ",col+12,"14 ");
            break;

        case 0x4000l :
            dispf("%*p%s ",col+12,"15 ");
            break;

        default :
            break;
    }
    break;      /* End Lead Number Switch. */
}              /* End Pod Identification Switch. */

if (event->status)
{
    dispf("%*p%s",col+17,"ON");
}
else
{
    dispf("%*p%s",col+17,"OFF");
}
}              /* End lead_change_disp( ) */

/*****
* The "lead_stat_disp" routine displays the status of interface pod leads.
*****/

lead_stat_disp(status,col,softkeys)
unsigned int status;
unsigned int col;
DECODE_KEYS softkeys[];

{
    dispf("\n%p",col+MARGIN);      /* Get a new line. */

    switch (softkeys[1].value)      /* Start of Pod Identification Switch. */
    {
        case 1 :      /* Start RS-232 and MIL-188C Interface Pod Switch. */
        {
            if (0x0001l & status)
            {
                dispf("%*p%s ",col,"RTS ");
            }
            else
            {
                dispf("%*pHs ",col,"RTS ");
            }
        }
    }
}

```

Program Listing - GBOPS_LEAD.csource (Continued)

```
if (0x0002l & status)
{
    dispf("%*p%s ",col+4,"CTS ");
}
else
{
    dispf("%*p%Hs ",col+4,"CTS ");
}

if (0x0004l & status)
{
    dispf("%*p%s ",col+8,"DSR ");
}
else
{
    dispf("%*p%Hs ",col+8,"DSR ");
}

if (0x0008l & status)
{
    dispf("%*p%s ",col+12,"DTR ");
}
else
{
    dispf("%*p%Hs ",col+12,"DTR ");
}

if (0x0010l & status)
{
    dispf("%*p%s ",col+16,"RI  ");
}
else
{
    dispf("%*p%Hs ",col+16,"RI  ");
}

if (0x0020l & status)
{
    dispf("%*p%s ",col+20,"CD  ");
}
else
{
    dispf("%*p%Hs ",col+20,"CD  ");
}
dispf("\n");

if (0x0040l & status)
{
    dispf("%*p%s ",col,"SQ  ");
}
else
{
    dispf("%*p%Hs ",col,"SQ  ");
}
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
    if (0x0080l & status)
    {
        dispf("%*p%s ",col+4,"DRS ");
    }
    else
    {
        dispf("%*p%Hs ",col+4,"DRS ");
    }

    if (0x0100l & status)
    {
        dispf("%*p%s ",col+8,"SRS ");
    }
    else
    {
        dispf("%*p%Hs ",col+8,"SRS ");
    }

    if (0x0200l & status)
    {
        dispf("%*p%s ",col+12,"SCS ");
    }
    else
    {
        dispf("%*p%Hs ",col+12,"SCS ");
    }

    if (0x0400l & status)
    {
        dispf("%*p%s ",col+16,"SCD ");
    }
    else
    {
        dispf("%*p%Hs ",col+16,"SCD ");
    }
}
break;      /* End RS-232 and MIL-188C Interface Pod Switch. */

case 2 :    /* Start RS-449 Interface Pod Switch. */
{
    if (0x0001l & status)
    {
        dispf("%*p%s ",col,"RS ");
    }
    else
    {
        dispf("%*p%Hs ",col,"RS ");
    }

    if (0x0002l & status)
    {
        dispf("%*p%s ",col+4,"CS ");
    }
    else
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
{
    dispf("%*p%Hs ",col+4,"CS ");
}

if (0x0004l & status)
{
    dispf("%*p%s ",col+8,"DM ");
}
else
{
    dispf("%*p%Hs ",col+8,"DM ");
}

if (0x0008l & status)
{
    dispf("%*p%s ",col+12,"TR ");
}
else
{
    dispf("%*p%Hs ",col+12,"TR ");
}

if (0x0010l & status)
{
    dispf("%*p%s ",col+16,"IC ");
}
else
{
    dispf("%*p%Hs ",col+16,"IC ");
}

if (0x0020l & status)
{
    dispf("%*p%s ",col+20,"RR ");
}
else
{
    dispf("%*p%Hs ",col+20,"RR ");
}

if (0x0040l & status)
{
    dispf("%*p%s ",col+24,"SQ ");
}
else
{
    dispf("%*p%Hs ",col+24,"SQ ");
}

dispf("\n");
if (0x0080l & status)
{
    dispf("%*p%s ",col,"SI ");
}
}
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
else
{
    dispf("%*p%Hs ",col,"SI ");
}

if (0x0100l & status)
{
    dispf("%*p%s ",col+4,"SRS ");
}
else
{
    dispf("%*p%Hs ",col+4,"SRS ");
}

if (0x0200l & status)
{
    dispf("%*p%s ",col+8,"SCS ");
}
else
{
    dispf("%*p%Hs ",col+8,"SCS ");
}

if (0x0400l & status)
{
    dispf("%*p%s ",col+12,"SRR ");
}
else
{
    dispf("%*p%Hs ",col+12,"SRR ");
}

if (0x0800l & status)
{
    dispf("%*p%s ",col+16,"IS ");
}
else
{
    dispf("%*p%Hs ",col+16,"IS ");
}

if (0x1000l & status)
{
    dispf("%*p%s ",col+20,"SF ");
}
else
{
    dispf("%*p%Hs ",col+20,"SF ");
}
if (0x2000l & status)
{
    dispf("%*p%s ",col+24,"RL ");
}
else
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
{
    dispf("%*p%Hs ",col+24,"RL ");
}

if (0x4000l & status)
{
    dispf("%*p%s ",col+28,"SS ");
}
else
{
    dispf("%*p%Hs ",col+28,"SS ");
}
}
break;      /* End RS-449 Interface Pod Switch. */

case 3 :      /* Start V.35 Interface Pod Switch. */
{
    if (0x0001l & status)
    {
        dispf("%*p%s ",col,"RS ");
    }
    else
    {
        dispf("%*p%Hs ",col,"RS ");
    }

    if (0x0002l & status)
    {
        dispf("%*p%s ",col+4,"CS ");
    }
    else
    {
        dispf("%*p%Hs ",col+4,"CS ");
    }

    if (0x0004l & status)
    {
        dispf("%*p%s ",col+8,"DSR ");
    }
    else
    {
        dispf("%*p%Hs ",col+8,"DSR ");
    }

    if (0x0008l & status)
    {
        dispf("%*p%s ",col+12,"DTR ");
    }
    else
    {
        dispf("%*p%Hs ",col+12,"DTR ");
    }

    if (0x0010l & status)
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
{
    dispf("%*p%s ",col+16,"RI ");
}
else
{
    dispf("%*p%Hs ",col+16,"RI ");
}

if (0x0020l & status)
{
    dispf("%*p%s ",col+20,"CD ");
}
else
{
    dispf("%*p%Hs ",col+20,"CD ");
}

if (0x0040l & status)
{
    dispf("%*p%s ",col+24,"LT ");
}
else
{
    dispf("%*p%Hs ",col+24,"LT ");
}
}
break;      /* End of V.35 Interface Pod Switch. */

default:    /* Start of Lead Only Switch. */
{
    if (0x0001l & status)
    {
        dispf("%*p%s ",col,"1 ");
    }
    else
    {
        dispf("%*p%Hs ",col,"1 ");
    }
}

    if (0x0002l & status)
    {
        dispf("%*p%s ",col+2,"2 ");
    }
    else
    {
        dispf("%*p%Hs ",col+2,"2 ");
    }
    if (0x0004l & status)
    {
        dispf("%*p%s ",col+4,"3 ");
    }
    else
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
{
    dispf("%*p%Hs ",col+4,"3 ");
}
if (0x0008l & status)
{
    dispf("%*p%s ",col+6,"4 ");
}
else
{
    dispf("%*p%Hs ",col+6,"4 ");
}

if (0x0010l & status)
{
    dispf("%*p%s ",col+8,"5 ");
}
else
{
    dispf("%*p%Hs ",col+8,"5 ");
}

if (0x0020l & status)
{
    dispf("%*p%s ",col+10,"6 ");
}
else
{
    dispf("%*p%Hs ",col+10,"6 ");
}

if (0x0040l & status)
{
    dispf("%*p%s ",col+12,"7 ");
}
else
{
    dispf("%*p%Hs ",col+12,"7 ");
}

if (0x0080l & status)
{
    dispf("%*p%s ",col+14,"8 ");
}
else
{
    dispf("%*p%Hs ",col+14,"8 ");
}
}
if (0x0100l & status)
{
    dispf("%*p%s ",col+16,"9 ");
}
else
```

Program Listing - GBOPS_LEAD.csource (Continued)

```
{
    dispf("%*p%Hs ",col+16,"9 ");
}

if (0x0200l & status)
{
    dispf("%*p%s ",col+18,"10 ");
}
else
{
    dispf("%*p%Hs ",col+18,"10 ");
}

if (0x0400l & status)
{
    dispf("%*p%s ",col+21,"11 ");
}
else
{
    dispf("%*p%Hs ",col+21,"11 ");
}

if (0x0800l & status)
{
    dispf("%*p%s ",col+24,"12 ");
}
else
{
    dispf("%*p%Hs ",col+24,"12 ");
}

if (0x1000l & status)
{
    dispf("%*p%s ",col+27,"13 ");
}
else
{
    dispf("%*p%Hs ",col+27,"13 ");
}

if (0x2000l & status)
{
    dispf("%*p%s ",col+30,"14 ");
}
else
{
    dispf("%*p%Hs ",col+30,"14 ");
}

if (0x4000l & status)
{
    dispf("%*p%s ",col+33,"15 ");
}
else
```

Program Listing - GBOPS_LEAD.csource (Continued)

```

        {
            dispf("%*p%Hs ",col+33,"15 ");
        }
    }
    break;        /* End of Lead Only Switch. */
}                /* End of Pod Identification Switch. */
}                /* End lead_stat_disp( ). */

```

Program Listing - USER.csource

```

/*****
*
*                               HP 18320A DataCommC
*                               Decoding Displays
*
*                               File:  USER.csource
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
*
* This user( ) routine creates the Decoding Displays screen header line and passes
* the location of the user decode routine to the display kernel.
*****/

extern DECODE_KEYS gbops_top[];    /* userkeys */
extern void main_dec();            /* userdec */
extern display();                  /* provided by HP, in main */

user()
{
    char *your_heading =
        "                               DCE                               ";

/*****
*
*                               Use this template to create an 80 character display line.
*
* "1234567890123456789012345678901234567890123456789012345678901234567890"
* "                               DCE                               "
*****/

    display(gbops_top,main_dec,your_heading);
}

```

Program Listing _ DECODE.include

```

/*****
*
*                               HP 18320A DataCommC
*                               Decoding Displays
*
*                               File: DECODE.include
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
*
*   This file defines the decode_keys structure.
*****/

/* key[].style values */
#define BLANK 0 /* Softkey is blank and hitting it does nothing. */
#define OFFON 1 /* Softkey is a toggle (half-bright or full). */
#define CYCLE 2 /* Stay on same level and cycle through the labels. */
#define ZCYCL 3 /* Same as CYCLE, but values start at 0. */
#define LAYER 4 /* Create a lower level of softkeys. */

typedef struct decode_keys
{
    int style;
    char *label[8];
    int value;
    struct decode_keys *next_layer;
}DECODE_KEYS;

```

Program Listing - MACROS.include

```

/*****
*
*                               HP 18320A DataCommC
*                               Decoding Displays
*
*                               File: MACROS.include
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
*
*   This file defines the Raw Bytes and Undecoded softkeys.
*****/

#define RAW_BYTES {ZCYCL,{'Raw Bytes',"One-Line Raw Bytes","All Raw Bytes"},{0}}
#define UNDECODED {ZCYCL,{'Undecoded',"One-Line Undecoded","All Undecoded"},{0}}

```

Program Listing - GBOP.include

```

/*****
*
*          HP 18320A DataCommC
*          Decoding Displays
*
*          File: GBOPS.include
*
*          Copyright 1989
*          Hewlett-Packard Company
*
*  The structures in this file define the frame and protocol.
*****/

#define UNDEF          -1

/* Control Field Format. */
#define SUPERV          1
#define INFO            2
#define UNUMB           3

/* Level 2 Protocol Types. */
#define LAPB            1
#define LAP              2
#define HDLC            3
#define SDLC            4

/* Control Field Identifiers. */
#define I                0      /* LAP, LAPB, SDLC, HDLC. */
#define RR               1      /* LAP, LAPB, SDLC, HDLC. */
#define RNR              2      /* LAP, LAPB, SDLC, HDLC. */
#define REJ              3      /* LAP, LAPB, SDLC, HDLC. */
#define SREJ             4      /* HDLC. */
#define SABME            5      /* LAPB, HDLC. */
#define SABM             6      /* LAPB, HDLC. */
#define DM_SARM          7      /* SARM (LAP, HDLC), DM (LAPB, SDLC, HDLC). */
#define DISC_RD          8      /* DISC (LAP, LAPB, HDLC, SDLC), RD(SDLC, HDLC). */
#define UA               9      /* LAP, LAPB, SDLC, HDLC. */
#define FRMR_CMDR       10      /* LAP, LAPB, SDLC, HDLC. */
#define CFGR            11      /* SDLC. */
#define UI              12      /* SDLC, HDLC. */
#define TEST            13      /* SDLC. */
#define SIM_RIM         14      /* SDLC, HDLC. */
#define SARME           15      /* HDLC. */
#define SNRME           16      /* HDLC. */
#define SNRM            17      /* SDLC, HDLC. */
#define UP              18      /* SDLC, HDLC. */
#define RSET            19      /* HDLC. */
#define BCN             20      /* SDLC. */
#define XID             21      /* SDLC, HDLC. */

```

Program Listing - GBOP.include (Continued)

```

/* Result Constants Definitions. */
#define OK          0          /* No error during decoding. */
#define INVALID     -1         /* Impossible to decode: length error. */
#define ADDR_ERR    0x01       /* Wrong value in address field. */
#define MULTI_ERR   0x02       /* Impossible to decode multi link information :
                                length error. */
#define FRMR_ERR    0x04       /* Impossible to decode FRMR information:
                                length error. */
#define UNEX_INFO   0x08       /* Unexpected information according to the kind of
                                frame. */

/* Rejected Frame: Level 2 information. */
typedef struct
{
    long int ctrl;          /* Control field of rejected frame. */
    int Vs;                 /* Value of V(s). */
    int Vr;                 /* Value of V(r). */
    BOOL is_command;        /* If TRUE, rejected frame is a command frame. */
    BOOL is_ctrl_undef;     /* Value of bit W. if TRUE:p/f_bit, ctrl_format Ns, Nr,
                                ctrl_id are unused. */
    BOOL is_ctrl_incor;     /* Value of bit X. */
    BOOL is_infoGTmax;      /* Value of bit Y. */
    BOOL is_Nr_invalid;     /* Value of bit Z. */
} FRMR_INFO;

/* LEX output. */
typedef struct
{
    long int result;        /* INVALID: error in length, only raw_lgth and raw_ptr are
                                defined. */
    int L2raw_lgth;         /* 0 if level2 frame cannot be reached (lead_event or error in
                                HP 4954A). */
    BYTE *L2raw_ptr;        /* NULL if not reached. */
    int fcs;                /* If FCS bad or abort, frame is not decoded*/

/* ADDRESS */
    int addr;              /* Value of address field. */
    BOOL is_command;       /* If not is a response. */
    BOOL is_multi;         /* Is it an address for multilink operation: if yes multilink
                                control fields are used. */

/* CONTROL */
    BOOL pf_bit;           /* Value of P/F bit;. */
    int ctrl_format;       /* UNUMB: Nr and Ns not used SUPERV: Nr used, Ns not used.
                                Other: Nr and Ns used. */
    int Ns;                /* Value of N(s). */
    int Nr;                /* Value of N(r). */
    int ctrl_id;           /* Identifier of control field. */

/* MULTILINK CONTROL */
    int MNs;               /* Multilink sequence number. */
    BOOL V_bit;            /* Void sequencing bit. */
    BOOL S_bit;            /* Sequence check option bit. */

```

Program Listing - GBOP.include (Continued)

```
    BOOL R_bit;          /* MLP Reset request bit. */
    BOOL C_bit;          /* MLP reset Confirmation bit. */
/*  INFO */
    FRMR_INFO frmr;      /* May be used if ctrl_id==FRMR. */
    int  L3raw_lgth;      /* 0 if not used. */
    BYTE *L3raw_ptr;      /* NULL if not used. */
    BOOL is_dceframe;     /* If NOT is DTE frame. */
} DEC_GBOPS;
```

Program Listing - GLOBUS.include

```

*****
*
*                               HP 18320A DataCommC
*                               Decoding Displays
*
*                               File: GLOBUS.include
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
* This file defines the data type, softkey constants, and the display constants.
*****/

/* Global decoder constants definitions. */

#define LINELGTH 40

/* SOFTKEYS constants definitions. */
#define NONE 0
#define HEXA 1
#define FULL 2
#define EBCDIC 3
#define BINARY 4

/* DISPLAY constants definitions. */
#define NT_COL 41 /* First column of the screen NETWORK. */
#define TE_COL 1 /* Column of the half screen TERMINAL. */
#define MARGIN 5 /* Width of margin reserved for protocol labels. */
#define INDENT1 7 /* First indentation on display. */
#define WARN_MARG 4 /* Warning margin: location of flag in case of warnings. */

/* Basic Types */
typedef int BOOL;
typedef unsigned char BYTE;
typedef struct
{
    int lgth;
    BYTE *ptr;
} RAW_STRING;

/* Level 3 Raw Frame */
typedef struct
{
    int L3raw_lgth;
    BYTE *L3raw_ptr;
    BOOL is_dceframe; /* If NOT is DTE frame. */
} L3RAW;

```


Transmitting a Frame

This program transmits, receives, processes, and displays user-created data frames by controlling the following.

- Data entry.
- DTE or DCE selection.
- Frame processing (e.g., data, format, source, definition, type, address, transmission, reception, decoding, display, etc.).
- Packet processing (e.g., transmission, reception, decoding, display, etc.).
- Analyzer Setup Menu (i.e., protocol, data code, parity, clock source, and bit rate).
- Protocol stack simulation.
- Event storage buffer.
- Message queuing.

Subsequent paragraphs describe the files, their general functions and uses, and how to run the programs. File listings are also furnished.

Frame Transmission Files

The following Solution disc files are used for transmitting, processing, and displaying frame data.

PROCESS1.csource	Defines equipment type, frame type, frame contents, and send the frame. It automatically defines the analyzer setup menu parameters and starts the other processes.
PROCESS2.csource	Receives frame data from the line, and displays it on the bottom half of the screen.
PROCESS3.csource	Displays packet data relayed by PROCESS2 on the top half of the screen.
PROCESS23.csource	Contains sub-routines used by both PROCESS2 and PROCESS3.

MAKE1.text	Creates PROCESS1.program file.
MAKE2.text	Creates PROCESS2.program file.
MAKE3.text	Creates PROCESS3.program file.
PROCESS2.linkcommand	An ASCII file used by MAKE2.text to create PROCESS2.program.
PROCESS3.linkcommand	An ASCII file used by MAKE3.text to create PROCESS3.program.
PROCESS_H.include	This file contains definitions for the transmission programs.

Running the Program

Running the program is a simple, two step operation.

1. Use the *MAKE1.text*, *MAKE2.text*, and *MAKE3.text* files to create the *PROCESS1.program*, *PROCESS2.program*, and *PROCESS3.program* files.
2. Start *PROGRAM1.program*.

MAKE.text Files

The *MAKE1.text*, *MAKE2.text*, and *MAKE3.text* files (see figures 4-1 through 4-3) have the following functions:

- Define the object files.
- Define the program name.
- Describe the sequence for running the MAKE file.

The format for these files is as follows.

- All entries start in column 1.
- Entries starting with a pound sign (#) are comments.
- Lines ending in a backslash (\) are continued on the following line.
- Lines starting with the dollar sign (\$) describe the execution sequence for making the program file.

See the *DataCommC User's Guide* for a complete description of the *MAKE* type file.

```
# Make for PROCESS1.csource
#

OBJECT=PROCESS1.object

PROGRAM=PROCESS1

$(PROGRAM).program $(OBJECT)
    link $(PROGRAM).linkcommand
```

Figure 4-1. MAKE1.text File Listing

```
# Make for PROCESS2.csource
#

OBJECT=PROCESS23.object PROCESS2.object

PROGRAM=PROCESS2

$(PROGRAM).program $(OBJECT)
    link $(PROGRAM).linkcommand
```

Figure 4-2. MAKE2.text File Listing

```
# Make for PROCESS3.csource
#

OBJECT=PROCESS23.object PROCESS3.object

PROGRAM=PROCESS3

$(PROGRAM).program $(OBJECT)
    link $(PROGRAM).linkcommand
```

Figure 4-3. MAKE3.text File Listing

Linkcommand Files

The ASCII link command files (see figures 4-4 and 4-5) define the paths to each object file called by MAKE2.text and MAKE3.text files.

```
c:C/HP/PROCESS23.object  
c:C/HP/PROCESS2.object
```

Figure 4-4. PROCESS2.linkcommand File

```
c:C/HP/PROCESS23.object  
c:C/HP/PROCESS3.object
```

Figure 4-5. PROCESS3.linkcommand File

Messages

Messages that can be generated when running this program include the following.

PROCESS1: Can't send a STOP message to PROGRAM3.
PROCESS1: Can't send EQUIP message to PROCESS2.
PROCESS1: Can't send QID message to PROCESS2.
PROCESS1: Can't send RX_COUNT message to PROGRAM2 (update_rxcount routine).
PROCESS1: Can't spawn PROCESS2.
PROCESS1: Can't spawn PROCESS3.
PROCESS1: Illegal bit name (get_PF_bit routine).
PROCESS1: Illegal control field type (unnumbered routine).
PROCESS1: Illegal equipment type (get_address routine).
PROCESS1: Illegal frame type (get_address routine).
PROCESS1: Received non-START message.
PROCESS1: Received non_RX_COUNT message from PROGRAM2 (update_rxcount routine).
PROCESS2: Can't read EQUIP message from PROCESS1.
PROCESS2: Can't read message queue.
PROCESS2: Can't read QID message from PROCESS1.
PROCESS2: Can't release the DataCommC event.
PROCESS2: Can't send packet message to PROCESS3.
PROCESS2: Can't send RXCOUNT message to PROCESS1.
PROCESS2: Can't send START message to PROCESS1.
PROCESS2: Can't send STOP message to PROCESS 3.
PROCESS2: Error in starting data acquisition.
PROCESS2: Error in stopping data acquisition.
PROCESS2: Non-DataCommC message received.
PROCESS2: Received non-EQUIP message.
PROCESS2: Received non-QID message.
PROCESS2: Undesired subtype on DataCommC message.
PROCESS3: Can't read message queue.
PROCESS3: Non-packet message received.

Source and Include File Listings

The following are Solutions disc source and include file listings. These listings have been reformatted for easier reading and the comments expanded to better describe function and operation. *The code has not been changed.* The code on the Solutions disc has been tested for proper operation prior to shipment.

As with all software products, the programs may be revised prior to shipment. Therefore, it is possible that there may be *minor* differences.

Program Listing - PROCESS1.csource

```

/*****
*
*                               HP 18320A DataCommC
*                               Transmitting a Frame
*
*                               File: PROCESS1.csource
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
*       This program allows the user to transmit DTE or DCE frames using softkeys.
*****/

#define PROCESS2 "c:C/HP/PROCESS2.program"
#define PROCESS3 "c:C/HP/PROCESS3.program"

#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <dlib.h>
#include <leads.h>
#include <message.h>
#include <video.h>
#include <retval.h>
#include "c:C/HP/PROCESS_H.h"

/* Global variables. */

int equip;      /* Equipment type: DTE_EQUIP or DCE_EQUIP. */
int numSent, numRx; /* N(s) and N(r) values. */
long process2_QID, process3_QID;
```

Program Listing - PROCESS1.csource (Continued)

```
/******
* The "initialize" routine clears the screen and prints a message.
*****/

void initialize()
{
    int illegal, key;

    wipe(TOP_ROW, BOTTOM_ROW);
    center("PROCESS1 can be either a DTE or DCE simulator...", MIDDLE_ROW);
    set_rows(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    center("Choose your equipment type", PROCESS1_COMMAND_LINE);
    put_all_sks("", "", "", "", "", "", "", "", "", (HALF_BRIGHT | INVERSE));
    put_all_sks(NULL, NULL, "DTE", NULL,
                NULL, "DCE", NULL, "MONITOR\ONLY", INVERSE);

    do      /* Loop until legitimate response made. */
    {
        illegal = 0;
        key = getch();

        if (key == SK3)
            equip = DTE_EQUIP;
        else
            if (key == SK6)
                equip = DCE_EQUIP;
            else
                if (key == SK8)
                    equip = MONITOR_EQUIP;
                else
                    illegal = 1;
    }

    while (illegal);

    numSent = numRx = 0;      /* Initialize N(s) and N(r) values. */
}
```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "configure" routine establishes the channel configuration and protocol.
*****/
void configure()
{
    if (equip == DTE_EQUIP)      /* DTE simulation. */
    {
        set_channelconfig(DCE, RX);
        set_channelconfig(DTE, TX);
    }
    else
    {
        if (equip == DCE_EQUIP)  /* DCE simulation. */
        {
            set_channelconfig(DTE, RX);
            set_channelconfig(DCE, TX);
        }
        else /* Monitor only. */
        {
            set_channelconfig(DTE, RX);
            set_channelconfig(DCE, RX);
        }
        set_bitrate(9600L);
        set_protocol(HDLC, SYNC_DCE, ASCII8, NO_PARITY, DEFAULT); /* Default values. */
        set_ignored_leads(IGNORE_ALL_LEADS);
    }
}

/*****
* The "spawn_others" routine starts Process 2 and Process 3.
*****/
void
spawn_others()
{
    char *sendargs2[ARGLIST_SIZE], *sendargs3[ARGLIST_SIZE];
    int retval;

    retval = spawn(PROCESS2, 0, sendargs2, PROCESS2_PRIO, &process2_QID);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS1: Can't spawn PROCESS2.", FATAL_ERROR);

    retval = spawn(PROCESS3, 0, sendargs3, PROCESS3_PRIO, &process3_QID);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS1: Can't spawn PROCESS3.", FATAL_ERROR);
}

```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "initialize_and_sync" routine initializes and synchronizes Process 2.
*****/
void
initial_and_sync()
{
    int retval;
    MESSAGE message, in_msg;
    message.type = EQUIP; /* Send equipment type to Process 2. */
    message.body.user_message.user_int1 = equip;
    retval = send_message(process2_QID, &message, ZERO);
    if (retval != SUCCESSFUL)
        error_notification("PROCESS1: Can't send EQUIP message to PROCESS2.",
                           FATAL_ERROR);
    message.type = QID; /* Send Process 3 QID to Process 2. */
    message.body.user_message.user_long = process3_QID;
    retval = send_message(process2_QID, &message, ZERO);

    do
    {
        /* Wait for Process 2 START message. */
        read_message(&in_msg, WAIT, WAIT_FOREVER);
        if (in_msg.type != START)
            error_notification("PROCESS1: Received non-START message.", NON_FATAL_ERROR);
    }
    while (in_msg.type != START);
}

if (retval != SUCCESSFUL)
    error_notification("PROCESS1: Can't send QID message to PROCESS2.",
                      FATAL_ERROR);

```

```

/*****
* The "packetize" routine places the user raw data string data in a Level 3 packet
* and then stores it in the character array "packet."
*
* PARAMETERS:
*   userdata - Pointer to a null-terminated string representing the raw
*               to be placed in the packet. The string is unchanged by the
*               routine.
*   packet   - An array of characters into which the null-terminated
*               packet is placed.
*
* NOTE:
*   PACKET_MAX defines the maximum number of characters stored in packet.
*****/
void
packetize(userdata, packet)
char *userdata;
char packet[];
{
    strncpy(packet, userdata, PACKET_MAX);
    packet[PACKET_MAX] = '\0'; /* Null-terminate the packet string. */
}

```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "sendframe" routine Takes the data packet "info," constructs a frame, and sends
* the frames using 'sendf.'
*
* PARAMETERS:
*   address - One octet that is the address to be assigned to the frame.
*   control - One octet that is the control information to be assigned to the
*             frame.
*   info    - A pointer to the information field character string. It is the
*             Level3 packet data.
*   fcs     - The FCS flag indicator. Possible values are: GOOD_FCS, BAD_FCS,
*             ABORT_FCS, and DEFAULT (i.e., GOOD_FCS). Any value constitutes an
*             error.
*
* RETURN VALUES:
*
*   SUCCESSFUL The frame was sent without error.
*   ERROR_1    Error: An error was found in the frame formatting.
*   ERROR_2    Error: The front-end cannot send this frame.
*   ERROR_3    Error: A send is already in progress.
*   ERROR_4    Error: Illegal frame type.
*   ERROR_5    Error: Front-end not configured for a send.
*   ERROR_6    Error: Send string >4106 or <2 bytes.
*****/

/*****
*   ERROR_7    Error: Invalid FCS type.
*   ERROR_10   FATAL ERROR while attempting to communicate with the front-end
*             hardware.
*****/

int
sendframe(address, control, info, fcs)
INT8 address;
INT8 control;
char *info;
char fcs;

{
    int result;
    if (fcs == DEFAULT)
        fcs = GOOD_FCS;
    if ((fcs == GOOD_FCS) || (fcs == BAD_FCS) || (fcs == ABORT_FCS))
    {
        result = sendf("%2n%s%f", address, control, info, fcs);
        return(result);
    }
    else /* Invalid FCS type. */
        return(ERROR_7);
}

```

Program Listing - PROCESS1.csource (Continued)

```
/*
*****
* The "get_command" routine uses the menu softkeys to choose the SUPERVISORY frame
* type as either COMMAND or RESPONSE. Press EXIT to leave the menu.
*****
*/
get_command_resp()
{
    int illegal, key, retval;
    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    center("Choose the frame type", PROCESS1_COMMAND_LINE);
    put_all_sks("", "", "", "", "", "", "", "", (HALF_BRIGHT | INVERSE));
    put_sk("COMMAND", SK3, INVERSE);
    put_sk("RESPONSE", SK6, INVERSE);
    put_sk("EXIT", SK8, INVERSE);

    do      /* Start valid key loop. */
    {
        illegal = 0;
        key = getch();

        switch (key)
        {
            /* Command and Response switch. */
            case SK3:
                retval = COMMAND;
                break;
            case SK6:
                retval = RESPONSE;
                break;
            case SK8:
                retval = EXIT;
                break;
            default:
                illegal = 1;
                break;
        }
        /* End Command and Response switch. */
    }
    /* End valid key loop. */
    while (illegal);
    return (retval);
}
```

Program Listing - PROCESS1.csource (Continued)

```

/*****
 * The "get_PF_bit" functions selects the frame P/F bit using the "bit_name" switch.  *
 * Press EXIT to leave the menu.  *
 *****/
get_PF_bit(bit_name)
int bit_name;
{
    int illegal, key, bit_val;
    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    put_all_sks("", "", "", "", "", "", "", "", (HALF_BRIGHT | INVERSE));

    if (bit_name == P_BIT)
    {
        center("Choose the P bit value", PROCESS1_COMMAND_LINE);
        put_sk("P BIT'CLEAR", SK3, INVERSE);
        put_sk("P BIT'SET", SK6, INVERSE);
        put_sk("EXIT", SK8, INVERSE);
    }
    else
        if (bit_name == F_BIT)
        {
            center("Choose the F bit value", PROCESS1_COMMAND_LINE);
            put_sk("F BIT'CLEAR", SK3, INVERSE);
            put_sk("F BIT'SET", SK6, INVERSE);
            put_sk("EXIT", SK8, INVERSE);
        }
    else
        error_notification("PROCESS1: Illegal bit name. (get_PF_bit routine).",
                           FATAL_ERROR);

    do      /* Softkey loop. */
    {
        illegal = 0;
        key = getch();

        switch (key)      /* Start "bit_name" switch. */
        {
            case SK3:
                bit_val = PF_CLEAR;
                break;
            case SK6:
                bit_val = PF_SET;
                break;
            case SK8:
                bit_val = EXIT;
                break;
            default:
                illegal = 1;
                break;
        }      /* End "bit_name" switch. */
    }      /* End Softkey loop. */
    while (illegal);
    return (bit_val);
}

```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "get_address" routine returns the frame address value based on frame type
* (COMMAND or RESPONSE) and equipment type (DCE or DTE).
*****/
get_address(type)
int type;
{
    int address;
    if (equip == DTE_EQUIP)
    {
        if (type == COMMAND)
            address = B_ADDRESS;
        else
            if (type == RESPONSE)
                address = A_ADDRESS;
            else
                error_notification("PROCESS1: Illegal frame type (get_address routine).",
                                   FATAL_ERROR);
    }
    else
        if (equip == DCE_EQUIP)
        {
            if (type == COMMAND)
                address = A_ADDRESS;
            else
                if (type == RESPONSE)
                    address = B_ADDRESS;
                else
                    error_notification("PROCESS1: Illegal frame type
                                       (get_address routine).",
                                       FATAL_ERROR);
        }
    else
        error_notification("PROCESS1: Illegal equipment type (get_address routine).",
                            FATAL_ERROR);
    return (address);
}

```

Program Listing - PROCESS1.csource (Continued)

```
/******
* The "get_string" routine enters the Level 3 I-frame character string.
******/
get_string(string, max_len)
char string[];
int max_len;
{
    int i, key, row, col;
    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    center("Type the Level 3 data string (and then <ENTER>):",
          PROCESS1_COMMAND_LINE);
    put_sk("EXIT", SK8, INVERSE);
    set_cursor(BOTTOM_ROW, 1);
    printf("=> ");
    i = 0;
    do
    {
        key = getch();
        if ((key == BACKSPACE) && (i > 0))
        {
            i--;
            string[i] = '\0';
            get_cursor(&row, &col);
            col--;
            set_cursor(row, col);          /* Move back one column. */
            putch((int) ' ');             /* Destroy what is there. */
            set_cursor(row, col);          /* Move back one column. */
        }
        else
        {
            if (isprint(key))
            {
                string[i] = (char) key;
                i++;
                putch(key);
            }
        }
    }
    while ((key != LINEFEED) && (key != SK8) && (i < max_len));
    string[i] = '\0';          /* NULL terminator. */
    if (key == SK8)
        return (EXIT);
    else
        return (SUCCESSFUL);
}
```

Program Listing - PROCESS1.csource (Continued)

```

/*****
 * The "update_rxcount" routine requests N(r) value from PROCESS 2.
 *****/
update_rxcount()
{
    int retval;
    MESSAGE message;

    message.type = RX_COUNT;
    retval = send_message(process2_QID, &message, ZERO);    /* Request N(r) value. */
    if (retval != SUCCESSFUL)
        error_notification("PROCESS1: Can't send RX_COUNT message\
                           to PROCESS2 (update_rxcount routine).", FATAL_ERROR);

    retval = read_message(&message, WAIT, WAIT_FOREVER);    /* Wait for N(r). */
    if (retval != SUCCESSFUL)
        error_notification("PROCESS1: Received non-RX_COUNT message \
                           from PROCESS2 (update_rxcount routine).", FATAL_ERROR);

    numRx = message.body.user_message.user_int1;            /* N(r) value. */
}

```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "unnumbered" routine controls sending an Unnumbered frame as follows.
*
* 1. Select the control field P/F bit value.
* For a SABM or DISC command frame, the menu asks for the P bit value.
* For a DM, UA, or FRMR response frame, the menu asks for the F bit value.
* The routine "get_PF_bit" uses the parameter P_BIT for SABM and DISC frames
* and the F_BIT parameter for the others. The routine returns one of the
* following values.
* 0 - Do not set the P/F bit.
* or
* an integer with only the P/F bit set. This indicates the P/F bit in the
* control field should be set by ORing the return value with the control
* field value.
* or
* EXIT to return to the previous menu.
d* 2. The address field value is chosen on the basis of the frame type (COMMAND or
*
* RESPONSE) and equipment type (DTE or DCE).
*
* 3. Send an Unnumbered frame with the selected address and control fields.
* The FRMR frame contains extra fields and is handled as a special case.
*****/

```

```

unnumbered(control)
int control;

{
    INT8 rej_d_cntl, state_vars, wxyz;
    int PF_bit, frame_rej, type, address;
    switch (control)
    {
        case SABM_FRAME:
        case DISC_FRAME:
            type = COMMAND;
            PF_bit = get_PF_bit(P_BIT);
            break;
        case DM_FRAME:
        case UA_FRAME:
        case FRMR_FRAME:
            type = RESPONSE;
            PF_bit = get_PF_bit(F_BIT);
            break;
        default:
            error_notification("PROCESS1: Illegal control field type
                               (unnumbered routine).",
                               FATAL_ERROR);
    }

    if (PF_bit != EXIT)
    {
        if (control == FRMR_FRAME)
            frame_rej = 1;
        else
            frame_rej = 0;
    }
}

```

Program Listing - PROCESS1.csource (Continued)

```
/* Returns the address field for the frame type. */
address = get_address(type);

control |= PF_bit;      /* Set P/F bit option. */

if (frame_rej)
{
    rej_d_cntl = DEF_REJD_CNTL;
    state_vars = (INT8) ((numSent << 4) | numRx);
    wxyz = DEF_WXYZ;
    sendf("%2n%3n", (INT8) address, (INT8) control,
          rej_d_cntl, state_vars, wxyz);
}
else
    sendf("%2n", (INT8) address, (INT8) control);
}      /* End of PF_bit 'if.' */

}      /* End of Unnumbered frame. */
```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "information" routine controls sending an Information frame as follows.
*
* 1. Sets P bit per user softkey input.
* 2. User-input Level 3 character string.
* 3. Get address field value based on equipment (DCE or DTE).
* 4. Set the I-frame control field (i.e., P bit value, N(s) and N(r) sub-fields.
* 5. Sends the Information frame with the selected address field, control field,
* and the Level 3 data.
*****/
information()
{
    char level3[PACKET_MAX + 1];
    int P_bit, retval;
    int address, control;

    do /* P-bit input loop. */
    {
        P_bit = get_PF_bit(P_BIT);

        if (P_bit != EXIT) /* When P-bit chosen, ask for string. */
            retval = get_string(level3, PACKET_MAX);
    }
    while ((P_bit != EXIT) && (retval == EXIT));

    if (P_bit != EXIT) /* When P-bit chosen and string entered. */
    {
        update_rxcount(); /* Update N(r) value. */

        address = get_address(COMMAND);
        control = I_FRAME | P_bit;

        /* Initialize control field value and set P bit. */
        control |= (numRx << 5); /* N(r) */
        control |= (numSent << 1); /* N(s) */

        sendf("%2n%s", (INT8) address, (INT8) control,
            strlen(level3), level3);

        numSent = (numSent + 1) % FC_MODULO;
    }
}

```

Program Listing - PROCESS1.csource (Continued)

```
/******
 * The "supervisory" routine controls sending a Supervisory frame as follows.
 *
 * 1. User softkey selects frame type (COMMAND or RESPONSE).
 * 2. User softkey inputs P-bit for COMMAND or F-bit for RESPONSE frame.
 * 3. Choose address field value based on equipment (DTE or DCE).
 * 4. Set up the control field P/F bit and N(r) sub-field.
 * 5. Send Supervisory frame with address and control fields.
 *****/

supervisory(control)
int control;
{
    int type, PF_bit, address;

    do      /* Select frame type and P/F-bit loop. */
    {
        type = get_command_resp();      /* Select frame type. */

        if (type == COMMAND)
            PF_bit = get_PF_bit(P_BIT);
        else
            if (type == RESPONSE)
                PF_bit = get_PF_bit(F_BIT);
    }
    while ((type != EXIT) && (PF_bit == EXIT));

    if (type != EXIT)      /* Type and P/F bit selected. */
    {
        update_rxcount();      /* Update N(r). */
        address = get_address(type);
        control |= PF_bit;      /* Set P/F bit. */
        control |= (numRx << 5);      /* N(r) value. */
        sendf("%2n", (INT8) address, (INT8) control);
    }
}
```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "menu_zero" routine is the QUIT menu.
*
* RETURN VALUE:
*   QUIT - User pressed QUIT softkey to halt the program.
*****/
menu_zero()
{
    int key;

    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    center("Press QUIT to halt program", PROCESS1_COMMAND_LINE);
    put_all_sks("", "", "", "", "", "", "", "", (INVERSE | HALF_BRIGHT));
    put_all_sks(NULL, NULL, NULL, NULL, NULL, NULL, NULL,
        "QUIT", INVERSE);

    do /* Start QUIT Loop. */
        key = getch();
    while (key != SK8);

    return (QUIT);
}

```

```

/*****
* The "menu_one" routine is the unnumbered frame menu.
*
* RETURN VALUES:
*   SUCCESSFUL - Frame sent or returned to top level menu.
*   OTHER_MENU - Select another top level menu.
*   QUIT - Halt the program.
*****/
menu_one()
{
    int retval, illegal, key;

    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    center("Choose a frame", PROCESS1_COMMAND_LINE);
    put_sk("", SK7, (HALF_BRIGHT | INVERSE));
    put_all_sks("SABM", "DISC", "UA", "DM", "FRMR", "OTHER FRAMES", NULL,
        "QUIT", INVERSE);
    retval = SUCCESSFUL;

    do /* Unnumbered frame loop. */
    {
        illegal = 0;
        key = getch();
        switch (key)
        {
            case SK1:
                unnumbered(SABM_FRAME);
                break;
            case SK2:
                unnumbered(DISC_FRAME);
                break;

```

Program Listing - PROCESS1.csource (Continued)

```
    case SK3:
        unnumbered(UA_FRAME);
        break;
    case SK4:
        unnumbered(DM_FRAME);
        break;
    case SK5:
        unnumbered(FRMR_FRAME);
        break;
    case SK6:
        retval = OTHER_MENU;
        break;
    case SK8:
        retval = QUIT;
        break;
    default:
        illegal = 1;
        break;
    }
}
while (illegal);

return (retval);
}
```

Program Listing - PROCESS1.csource (Continued)

```

/*****
* The "menu_two" routine is the Information frame and Supervisory frame menu.
*
* RETURN VALUES;
*   SUCCESSFUL - Frame sent or returned to top level menu.
*   OTHER_MENU - Select another top level menu.
*   QUIT       - Halt the program.
*****/
menu_two()
{
    int retval, illegal, key;

    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);
    center("Choose a frame", PROCESS1_COMMAND_LINE);
    put_sk("", SK5, (HALF_BRIGHT | INVERSE));
    put_sk("", SK7, (HALF_BRIGHT | INVERSE));
    put_all_sks("I'(info)", "RR", "RNR", "REJ", NULL, "OTHER'FRAMES", NULL,
               "QUIT", INVERSE);
    retval = SUCCESSFUL; /* may change in switch statement */

    do /* Information and Supervisory frame loop. */
    {
        illegal = 0;
        key = getch();

        switch (key)
        {
            case SK1:
                information();
                break;
            case SK2:
                supervisory(RR_FRAME);
                break;
            case SK3:
                supervisory(RNR_FRAME);
                break;
            case SK4:
                supervisory(REJ_FRAME);
                break;
            case SK6:
                retval = OTHER_MENU;
                break;
            case SK8:
                retval = QUIT;
                break;
            default:
                illegal = 1;
                break;
        }
    }
    while (illegal);

    return (retval);
}

```

Program Listing - PROCESS1.csource (Continued)

```
/******
 * The "main" routine starts Process1.
 *****/
main()
{
    char key;
    char packet[PACKET_MAX+1];
    MESSAGE message, in_msg;
    int retval, retval_one, retval_two;

    initialize();
    configure();           /* Configure the front-end. */
    spawn_others();        /* Start Process 2 and Process 3. */
    initial_and_sync();     /* Synchronize with Process 2. */
    retval_two = OTHER_MENU;
    if (equip != MONITOR_EQUIP)

    do      /* Send frames loop. */
    {
        if (retval_two == OTHER_MENU)
            retval_one = menu_one();

        if (retval_one == OTHER_MENU)
            retval_two = menu_two();

    }
    while ((retval_one != QUIT) && (retval_two != QUIT));

    else      /* Monitor mode; do not send frames. */
        menu_zero();

    wipe(PROCESS1_COMMAND_LINE, BOTTOM_ROW);    /* Clear the softkeys. */

    message.type = STOP;      /* Send STOP message to Process 2. */
    retval = send_message(process2_QID, &message, ZERO);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS1: Can't send STOP message to PROCESS2.",
        FATAL_ERROR);
    exit();
}
```

Program Listing - PROCESS2.csource

```

/*****
*
*          HP 18320A DataCommC
*          Transmitting a Frame
*
*          File:  PROCESS2.csource
*
*          Copyright 1989
*          Hewlett-Packard Company
*
* This program receives DTE frame-level data (PROCESS1) and DCE frame-level data.
* The data is displayed on the BOTTOM half of the screen in frame-level fields.  DTE
* data is displayed in normal video and DCE data is displayed in inverse video.
*
* After frame-level data are displayed, packet level data (i.e., the I-field and
* packet size in bytes) are sent to PROCESS3.
*
* When PROCESS1 places a STOP message in the PROCESS2 queue, PROCESS2 sends a STOP
* message to PROCESS3, calls "stop_data" to halt data acquisition, exits.
*****/
/* To print non-fatal error messages, define VERBOSE. */
/* #define VERBOSE */

#include <stdio.h>
#include <stdlib.h>
#include <dlib.h>
#include <message.h>
#include <retval.h>
#include <video.h>

#include "c:C/HP/PROCESS_H.include"

* PROCESS2 Global Variables. */
int numRx;
int equip;
extern long _parent_QID;

long process3_QID;
MESSAGE message;

/*****
* The "initial_and_sync" routine performs initialization in coordination with PROCESS1
* as follows.
*
* 1. Sets up display on the bottom half of the screen.
* 2. Waits for PROCESS1 to define the equipment type (DTE or DCE).
* 3. Waits for PROCESS1 to define PROCESS3 QID.
* 4. Extracts the PROCESS3 QID (process3_QID).
* 5. Starts data acquisition by calling "start_data."
* 6. Tells PROCESS1 to start sending data by transmitting a START message.
*****/

```

Program Listing - PROCESS2.csource (Continued)

```

void
initial_and_sync()
{
    char key;
    int retval;
    MESSAGE to_go_msg;

    /* Select the bottom half of the screen. */
    setscreen(BOTTOM_HALF,
               "Address Control      Control Expansion      Other      Type
               FCS");

    numRx = 0;      /* Initialize N(r) value. */

    /* Wait for the equipment type (DTE or DCE) from PROCESS1. */
    retval = read_message(&message, WAIT, WAIT_FOREVER);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS2: Can't read EQUIP message from PROCESS1.",
                           FATAL_ERROR);

    if (message.type != EQUIP)
        error_notification("PROCESS2: Received non-EQUIP message.", FATAL_ERROR);

    equip = message.body.user_message.user_int1;    /* Equipment type. */

    /* Wait for PROCESS1 to identify PROCESS3 QID. */
    retval = read_message(&message, WAIT, WAIT_FOREVER);
    if (retval != SUCCESSFUL)
        error_notification("PROCESS2: Can't read QID message from PROCESS1.",
                           FATAL_ERROR);

    if (message.type != QID)
        error_notification("PROCESS2: Received non-QID message.", FATAL_ERROR);

    process3_QID = message.body.user_message.user_long;    /* PROCESS3 QID */

    /* Begin Data Acquisition. */
    retval = start_data();    /* Start the front-end. */
    if (retval != SUCCESSFUL)
        error_notification("PROCESS2: Error in starting the data acquisition.",
                           FATAL_ERROR);

    /* Send START message to PROCESS1. */
    to_go_msg.type = START;
    retval = send_message(_parent_QID, &to_go_msg, ZERO);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS2: Can't send START message to PROCESS1.",
                           FATAL_ERROR);
}

```

Program Listing - PROCESS2.csource (Continued)

```
/******
* The "extract_frame" routine extracts field values from the frame and returns them *
* through the parameters listed below. *
* *
* PARAMETERS: *
*   frame - Pointer the string representing the frame. The routine does not *
*           change the string. *
*   f_length - The frame length in bytes. *
*   address - Pointer that returns the address field value. *
*             Call overwrites previous value at location address. *
*   control - Pointer that returns the control field value. *
*             Call overwrites previous value at location control. *
*   packet - Pointer to the character array where the packet (Level 3) data is *
*            stored. *
*   p_length - Pointer that returns the number of bytes in the packet data. *
*              p_length = 0 for Supervisory and Unnumbered frames. (The Unnumbered *
*              frame FRMR is an exception in that it is a non-zero.) *
*****/
void
extractframe(frame, f_length, address, control, packet, p_length)
char *frame;
int f_length;
int *address;
int *control;
char packet[];
int *p_length;
{
    *address = (int) *frame++; /* Address field. */
    *control = (int) *frame++; /* Control field. */
    *p_length = f_length - 2;
    strncpy(packet, frame, *p_length);
    packet[*p_length] = '\0'; /* Add NULL terminator. */
}
```

Program Listing - PROCESS2.csource (Continued)

```

/*****
* The "get_frame_type" routines returns the frame type, either COMMAND or RESPONSE,
* for the user-defined address and source.
*
* PARAMETERS:
*     address -   Frame address field value.
*     source   -   Frame source; either DTE FRAME or DCE FRAME.
*
*
* RETURN VALUES:
*     COMMAND -   Frame issued as a command.
*     RESPONSE -  Frames issued as a response.
*****/
get_frame_type(address, source)
int address;
int source;

{
    int frame_type;

    if (address == A_ADDRESS)
    {
        if (source == DTE_FRAME)
            frame_type = RESPONSE;
        else
            if (source == DCE_FRAME)
                frame_type = COMMAND;
            else
                frame_type = UNKNOWN;
    }
    else
        if (address == B_ADDRESS)
        {
            if (source == DTE_FRAME)
                frame_type = COMMAND;
            else
                if (source == DCE_FRAME)
                    frame_type = RESPONSE;
                else
                    frame_type = UNKNOWN;
        }
    else
        frame_type = UNKNOWN;
    return (frame_type);
}

```

Program Listing - PROCESS2.csource (Continued)

```

/*****
* The "print_address" routine displays the Level 2 frame address.
*****/
void
print_address(address)
int address;
{
    address &= 0xff; /* Mask the high address bits. */
    printf("0x%02x ", address);

    if (address == A_ADDRESS)
        printf("(A) ");
    else
        if (address == B_ADDRESS)
            printf("(B) ");
        else
            printf("??? ");
}

/*****
* The "print_control" routine displays the Level 2 frame control and returns the
* frame name.
*
* Valid frame names are:
*
*     DISC_FRAME    DM_FRAME    FRMR_FRAME    I_FRAME    REJ_FRAME
*     RNR_FRAME     RR_FRAME     SABM_FRAME    UA_FRAME
*****/
print_control(control)
int control;
{
    int bottom_two_bits, frame_name;
    control &= 0xff; /* Mask the high order bits. */
    printf("0x%02x ", control);
    bottom_two_bits = control & 0x03;
    if (bottom_two_bits == SUPER_FRAME) /* Supervisory Frame. */
    {
        frame_name = control & BOTTOM_HALF; /* Mask P/F bit and N(r). */

        switch (frame_name) /* Start frame name switch. */
        {
            case RR_FRAME:
                printf("RR      ");
                break;
            case RNR_FRAME:
                printf("RNR    ");
                break;
            case REJ_FRAME:
                printf("REJ    ");
                break;
            default:
                printf("???    ");
                break;
        }
    }
}

```

Program Listing - PROCESS2.csource (Continued)

```
    } /* End frame name switch. */
}
else
    if (bottom_two_bits == UNNUM) /* Unnumbered frame. */
    {
        frame_name = control & (~PF_SET); /* Mask P/F bit. */
        switch (frame_name) /* Start frame name switch. */
        {
            case SABM_FRAME:
                printf("SABM ");
                break;
            case DM_FRAME:
                printf("DM ");
                break;
            case DISC_FRAME:
                printf("DISC ");
                break;
            case UA_FRAME:
                printf("UA ");
                break;
            case FRMR_FRAME:
                printf("FRMR ");
                break;
            default:
                printf("???? ");
                break;
        } /* End frame name switch. */
    }
else /* Information frame. */
{
    frame_name = I_FRAME;
    printf("I (info) ");
}
return (frame_name);
}
```

Program Listing - PROCESS2.csource (Continued)

```

/*****
 * The "print_PF" routine displays the bit name and the control.
 *****/
void
print_PF(bit_name, control)
int bit_name;
int control;
{
    int bit_val;

    bit_val = control & PF_SET;    /* Mask P/F bit. */

    if (bit_name == P_BIT)
    {
        if (bit_val)
            printf("P=1 ");
        else
            printf("P=0 ");
    }
    else
        if (bit_name == F_BIT)
        {
            if (bit_val)
                printf("F=1 ");
            else
                printf("F=0 ");
        }
}

/*****
 * The "expand_control" routine displays the following parameters.
 *
 * PARAMETERS:
 *   frame_name - Either DISC_FRAME, DM_FRAME, FRMR_FRAME, I_FRAME,
 *                 REJ_FRAME, RNR_FRAME, RR_FRAME, SABM_FRAME, or UA_FRAME.
 *   address    - Original address field value.
 *   control    - Original control field value.
 *   source     - Frame source; either DTE_FRAME or DCE_FRAME.
 *****/
void
expand_control(frame_name, address, control, source)
int frame_name;
int address;
int control;
int source;
{
    int frame_type;

    switch (frame_name)    /* Start frame name switch. */

```

Program Listing - PROCESS2.csource (Continued)

```
(
    case I_FRAME:
        printf("N(S)=%3d ", (int) ((control >> 1) & 0x07));
        print_PF(P_BIT, control);
        printf("N(R)=%3d ", (int) ((control >> 5) & 0x07));
        break;
    case RR_FRAME:
    case RNR_FRAME:
    case REJ_FRAME:
        printf("      ");
        frame_type = get_frame_type(address, source);
        if (frame_type == COMMAND)
            print_PF(P_BIT, control);
        else
            if (frame_type == RESPONSE)
                print_PF(F_BIT, control);
            else
                printf("      "); /* number of spaces used by print_PF */
        printf("N(R)=%3d ", (int) ((control >> 5) & 0x07));
        break;
    case SABM_FRAME:
    case DISC_FRAME:
        printf("      ");
        print_PF(P_BIT, control);
        printf("      ");
        break;
    case DM_FRAME:
    case UA_FRAME:
    case FRMR_FRAME:
        printf("      ");
        print_PF(F_BIT, control);
        printf("      ");
        break;
    default:
        printf("      ");
        break;
} /* End of frame switch. */
```

Program Listing - PROCESS2.csource (Continued)

```

/*****
 * The "print_extras" routine prints the two types of special data.
 *
 * 1. Prints three bytes of rejection data for FRMR_FRAME.
 * 2. Prints Level 3 data information for I_FRAME.
 *****/
void
print_extras(frame_name, packet, p_length)
int frame_name;
char *packet;
int p_length;
{
    if (frame_name == FRMR_FRAME)
    {
        printf("0x%02x 0x%02x 0x%02x   ", (int) ((*packet) & 0xff),
              (int) (*(packet + 1) & 0xff),
              (int) (*(packet + 2) & 0xff));
    }
    else
        if (frame_name == I_FRAME)
        {
            printf("%03d bytes Level 3   ", p_length);
        }
    else
        printf("               ");
}

/*****
 * The "print_type" routine prints the frame type for a frame with the user-defined
 * address and source.
 * PARAMETERS:
 *   address - Original address field value.
 *   source  - Frame source; either DTE_FRAME or DCE_FRAME.
 *****/
void
print_type(address, source)
int address;
int source;
{
    int frame_type;
    frame_type = get_frame_type(address, source);
    if (frame_type == COMMAND)
        printf("COMMAND ");
    else
        if (frame_type == RESPONSE)
            printf("RESPONSE ");
        else
            printf("UNKNOWN ");
}

```

Program Listing - PROCESS2.csource (Continued)

```

/*****
 * The "print_fcs" routine displays the FCS for the user-selected frame.
 *****/
void
print_fcs(fcs)
int fcs;
{
    switch (fcs)      /* Start FCS switch. */
    {
        case GOOD_FCS:
            printf("GOOD");
            break;
        case BAD_FCS:
            printf("BAD ");
            break;
        case ABORT_FCS:
            printf("ABRT");
            break;
        default:
            printf("????");
            break;
    }
    /* End FCS switch. */
}

```

Program Listing - PROCESS2.csource (Continued)

```

/*****
* The "dispframe" routine displays a formatted frame on one line. DTE frames
* are displayed in normal video; DCE frames are displayed in inverse video.
*
* The fields are passed in through the following parameters.
*
* PARAMETERS:
*   source - Frame source; either DTE_FRAME or DCE_FRAME.
*   address - Address field value.
*   control - Control field value.
*   packet - Pointer to the character array where the packet (Level 3)
*           data is stored.
*   p_length - The number of bytes in the packet..
*   fcs - The frame check sequence type (i.e., GOOD_FCS, BAD_FCS, or
*        ABORT_FCS).
*****/
dispframe(source, address, control, packet, p_length, fcs)
int source;
int address;
int control;
char packet[];
int p_length;
char fcs;
{
    int frame_name;
    if (source == DCE_FRAME)
        set_attribute(INVERSE);      /* Set inverse video for DCE frame. */
    else
        if (source != DTE_FRAME)
        {
            printf("dispframe: Frame source unknown (not DTE or DCE)\n");
            return;
        }
    print_address(address);
    frame_name = print_control(control);
    expand_control(frame_name, address, control, source);
    print_extras(frame_name, packet, p_length);
    print_type(address, source);
    print_fcs(fcs);
    printf("\n");

    set_attribute(NORMAL);          /* Set normal video for DTE. */
    return(frame_name);
}

```

Program Listing - PROCESS2.csource (Continued)

```
/* *****
 * The "main" routine reads and displays DataCommC messages from the queue until a
 * PROCESS1 STOP message is received.
 * ***** */
main()
{
    char key, fcs, *frame, packet[PACKET_MAX];
    INT8 type;
    int address, control;
    int subtype;
    int retval, f_length, p_length, frame_name;
    FRAME_EVENT *frame_event;
    MESSAGE p_message, message2;

    initial_and_sync(); /* Coordinate with PROCESS1 and start data acquisition. */

    retval = read_message(&message, WAIT, WAIT_FOREVER);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS2: Can't read message queue.", FATAL_ERROR);

    type = message.type;

    while (type != STOP) /* Process messages until STOP message received. */
    {
        if (type == *DATACOMM) /* Process DataCommC messages. */
        {
            subtype = (int) message.subtype;

            if ((subtype == DTE_FRAME) || (subtype == DCE_FRAME))
            {
                /* Frame event pointer. */
                frame_event = (FRAME_EVENT *) message.body.event.event_ptr;
                frame = (char *) frame_event->frame; /* Frame data pointer. */
                f_length = frame_event->length; /* Number of bytes. */
                fcs = frame_event->fcs;
                extractframe(frame, f_length, &address, &control,
                            packet, &p_length);
                frame_name = dispframe(subtype, address, control,
                                      packet, p_length, fcs);
                if (frame_name == I_FRAME)
                {
                    /* If I-frame not sent by PROCESS1, update numRx. */
                    if (((equip == DTE_EQUIP) && (subtype == DCE_FRAME)) ||
                        ((equip == DCE_EQUIP) && (subtype == DTE_FRAME)))
                        numRx = (numRx + 1) % FC_MODULO;

                    /* Prepare to send packet data to PROCESS3. */
                    p_message.type = (INT8) subtype; /* DTE_FRAME or DCE_FRAME. */

                    /* Pointer to packet. */
                    p_message.body.user_message.user_long = (INT32) packet;
                }
            }
        }
    }
}
```

Program Listing - PROCESS2.csource (Continued)

```
        /* Number of bytes in packet. */
        p_message.body.user_message.user_int1 = (INT16) p_length;

        retval = send_message(process3_QID, &p_message, ZERO);

        if (retval != SUCCESSFUL)
            error_notification("PROCESS2: Can't send packet message\
                                to PROCESS3.", FATAL_ERROR);
    }
}
else /* Neither a DTE_FRAME or a DCE_FRAME. */
{
#ifdef VERBOSE
    error_notification("PROCESS2: Undesired subtype on DataCommC message.",
                        NON_FATAL_ERROR);
#endif

}

/* Release the event to free buffer space. */
retval = release_event(message.body.event.event_ptr);

if (retval != SUCCESSFUL)
    error_notification("PROCESS2: Can't release DataCommC event.",
                        FATAL_ERROR);
} /* End processing DataCommC messages. */
else
if (type == RX_COUNT) /* PROCESS1 is asking for the N(r) value. */
{
    message2.type = RX_COUNT;
    message2.body.user_message.user_int1 = numRx;
    retval = send_message(_parent_QID, &message2, ZERO);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS2: Can't send RXCOUNT message to PROCESS1.",
                            FATAL_ERROR);
}
else /* Skip messages that are not DataCommC or RXCOUNT. */
{
#ifdef VERBOSE
    error_notification("PROCESS2: Non-DataCommC message received.",
                        NON_FATAL_ERROR);
#endif
}
/* Get the next message. */
retval = read_message(&message, WAIT, WAIT_FOREVER);
if (retval != SUCCESSFUL)
    error_notification("PROCESS2: Can't read message queue.", FATAL_ERROR);
```

Program Listing - PROCESS2.csource (Continued)

```
    type = message.type;
}    /* End of the while. */

    /* STOP message encountered. */
    retval = send_message(process3_QID, &message, ZERO);

/* Send STOP message to PROCESS3. */
if (retval != SUCCESSFUL)
    error_notification("PROCESS2: Can't send STOP message to PROCESS3.",
                      NOW_FATAL_ERROR);
    retval = stop_data(); /* Stop the front-end. */
if (retval != SUCCESSFUL)
    error_notification("PROCESS2: Error in stopping the data acquisition.",
                      NOW_FATAL_ERROR);

    exit();
}
```

Program Listing - PROCESS3.csource

```
/******
*                               HP 18320A DataCommC                               *
*                               Transmitting a Frame                               *
*                                                                           *
*                               File: PROCESS3.csource                           *
*                                                                           *
*                               Copyright 1989                                   *
*                               Hewlett-Packard Company                           *
*                                                                           *
*                               *
*                               *
*    This program receives packet-level data from PROCESS2 and displays it in TOP *
*    half of the screen. DTE data is displayed in normal video and DCE data is   *
*    displayed in inverse video.                                                 *
******/
#include <stdio.h>
#include <stdlib.h>
#include <dlib.h>
#include <message.h>
#include <retval.h>
#include <video.h>

#include "c:C/HP/PROCESS_H.Include"

void
initialize()    /* Initializes the screen. */
{
    setscreen(TOP_HALF, "Level 3 Raw Data");
}
```

Program Listing - PROCESS3.csource (Continued)

```
/******
 * The "extractpacket" routine extracts fields from a packet based on the following
 * parameter values.
 *
 * PARAMETERS:
 *   packet   - A pointer to the null-terminated string representing the Level 3
 *              packet data. The routine does not change this string.
 *   size     - The number of characters in the packet string. The null-terminator
 *              is not counted.
 *   userdata - A pointer to the character array to which packet data is copied.
 *
 *   u_length - An integer pointer that holds the number of user data bytes upon
 *              return from the call.
 *****/
void
char *packet;
int size;
char userdata[];
int *u_length;
{
    if (size > 0)
    {
        strcpy(userdata, packet);
        *u_length = size;
    }
    else
    {
        *userdata = '\0';      /* Empty string. */
        *u_length = 0;
    }
}
```

Program Listing - PROCESS3.csource (Continued)

```

/*****
 * The "disppacket" routine displays a formatted packet on one line of the display.
 * DTE packets are displayed in normal video and DCE packets are displayed in inverse
 * video. The following parameter values define the packet fields.
 *
 * PARAMETERS:
 *   type      - The frame source; either DTE_FRAME or DCE_FRAME.
 *   userdata  - A pointer to character array containing the user data.
 *   u_length  - The number of user data field bytes.
 *****/
void
disppacket(type, userdata, u_length)
INT8 type;
char *userdata;
int u_length;

{
    char line[USER_WIDTH+1];
    int i;
    if (type == DCE_FRAME)
        set_attribute(INVERSE);      /* Set inverse video for DCE_FRAME. */
    else
        if (type != DTE_FRAME)
        {
            printf("disppacket: Packet source unknown (not DTE or DCE)\n");
            return;
        }
    i = 0;

    /* Loop until there is no more user data or 80 characters are copied. */
    while ((*userdata) && (i < USER_WIDTH))
        line[i++] = *userdata++;      /* Copy user data one character at a time. */

    for ( ; i < USER_WIDTH; i++)
        line[i] = ' ';               /* Pad the display line with spaces. */

    line[i] = '\0';                  /* Add a null-terminator. */
    printf("%s\n", line);            /* Print the line. */
    set_attribute(NORMAL);           /* Set normal video for DTE_FRAME. */
}

```

Program Listing - PROCESS3.csource (Continued)

```
/******
 * The "main" routine reads PROCESS2 messages from the queue until a STOP message is *
 * received. *
 *****/
main()*
{
    char key, *packet;
    char userdata[PACKET_MAX];
    int retval, p_length, u_length;
    INT8 type;
    MESSAGE message;

    initialize(); /* miscellaneous initializations */
    retval = read_message(&message, WAIT, WAIT_FOREVER);

    if (retval != SUCCESSFUL)
        error_notification("PROCESS3: Can't read message queue.", FATAL_ERROR);

    type = message.type;

    while (type != STOP)
    {
        /* Process only DTE_FRAME or DCE_FRAME messages. */
        if ((type != DTE_FRAME) && (type != DCE_FRAME))
        {
            error_notification("PROCESS3: Non-packet message received.",
                               NON_FATAL_ERROR);
        }
        else /* Process packet message. */
        {
            packet = (char *) message.body.user_message.user_long; /* Packet pointer. */
            /* Number of packet bytes. */
            p_length = (int) message.body.user_message.user_int1;
            extractpacket(packet, p_length, userdata, &u_length);
            disppacket(type, userdata, u_length);
        } /* End of packet message processing. */

        /* Get the next message. */
        retval = read_message(&message, WAIT, WAIT_FOREVER);

        if (retval != SUCCESSFUL)
            error_notification("PROCESS3: Can't read message queue.",
                               FATAL_ERROR);

        type = message.type;
    } /* End DTE_FRAME or DCE_FRAME message while. */
    /* STOP message received. */
    exit();
}
```

Program Listing - PROCESS23.csource

```

/*****
*
*      HP 18320A DataCommC
*      Transmitting a Frame
*
*      File:  PROCESS23.csource
*
*      Copyright 1989
*      Hewlett-Packard Company
*
*  This program controls the analyzer screen for use by PROCESS2 and PROCESS3.
*****/
#define DEBUG3

#include <retval.h>
#include <video.h>
#include "c:C/HP/PROCESS_H.include"

/*****
*  The "setscreen" routine allocates either the top half or the bottom half of the
*  display screen to a calling process that can share the display.
*
*  PARAMETERS:
*      half      - TOP_HALF or BOTTOM_HALF are the only values this routine allows.
*      header    - A pointer to the null-terminated string that is displayed as the
*                  header of either the top or bottom half of the screen.
*
*  RETURN VALUES:
*      SUCCESSFUL - Either TOP_HALF or BOTTOM_HALF selected.
*      ERROR_1   - Neither TOP_HALF nor BOTTOM_HALF selected.
*      SIDE EFFECT - The routine draws a horizontal line across the middle of the
*                    screen when TOP_HALF selected.
*
*  CAUTION:
*      There is nothing to prevent two processes from selecting the SAME half of the
*      screen (i.e., no mutual exclusion is enforced).
*****/
int
setscreen(half, header)
int half;
char *header;
{
    if (half == TOP_HALF)
    {
        set_rows(TOP_FIRST_ROW-1, TOP_FIRST_ROW);
        wipe(TOP_FIRST_ROW-1, TOP_FIRST_ROW);
        set_cursor(TOP_FIRST_ROW-1, 1);
        set_attribute(HALF_BRIGHT);
        printf("%s", header);
        set_attribute(NORMAL);
        set_rows(MIDDLE_ROW-1, MIDDLE_ROW);
        set_cursor(MIDDLE_ROW, 1);
        printf("-----");
    }
}

```

Program Listing - PROGRAM23.csource

```
    printf("-----");
    set_rows(TOP_FIRST_ROW, TOP_LAST_ROW);
    wipe(TOP_FIRST_ROW, TOP_LAST_ROW);
    set_cursor(TOP_FIRST_ROW, 1);
    return(SUCCESSFUL);
}
else
if (half == BOTTOM_HALF)
{
    set_rows(BOT_FIRST_ROW-1, BOT_FIRST_ROW);
    wipe(BOT_FIRST_ROW-1, BOT_FIRST_ROW);
    set_cursor(BOT_FIRST_ROW-1, 1);
    set_attribute(HALF_BRIGHT);
    printf("%s", header);
    set_attribute(NORMAL);
    set_rows(BOT_FIRST_ROW, BOT_LAST_ROW);
    wipe(BOT_FIRST_ROW, BOT_LAST_ROW);
    set_cursor(BOT_FIRST_ROW, 1);
    return(SUCCESSFUL);
}
else /* Incorrect "half" parameter entered. */
return(ERROR_1);
}
```

Program Listing - PROCESS_H.include

```
/******
*
*                               HP 18320A DataComm
*                               Decoding Displays
*
*                               File: PROCESS_H.include
*
*                               Copyright 1989
*                               Hewlett-Packard Company
*
*
*   This file contains the frame and protocol determination routines.
*
******
/* Screen Constants Definitions. */

#define TOP_ROW 1      /* top row of entire screen */

#define MIDDLE_ROW 11

#define BOTTOM_ROW 25   /* bottom row of entire screen */

/* Constants passed to the 'setscreen' routine */

#define TOP_HALF 0
#define BOTTOM_HALF 1
```

Program Listing - PROCESS_H.Include

```
#define TOP_FIRST_ROW 2      /* first row of top half of screen */
#define TOP_LAST_ROW 10     /* last row of top half */
#define BOT_FIRST_ROW 13    /* first row of bottom half */
#define BOT_LAST_ROW 21     /* last row of bottom half */
#define PSOS1_COMMAND_LINE 22 /* Softkey menu prompt line. */

/* Equipment Type Definitions. */
#define DTE_EQUIP 0x00
#define DCE_EQUIP 0x01
#define MONITOR_EQUIP 0x02

/* Additional Message Types. */
#define QID 20
#define START 21
#define STOP 22
#define RX_COUNT 23
#define EQUIP 24

/* Frame Address Field Values. */
#define A_ADDRESS 0x03
#define B_ADDRESS 0x01

/* Frame Control Field Base Values. */
#define I_FRAME 0x00
#define RR_FRAME 0x01
#define RNR_FRAME 0x05
#define REJ_FRAME 0x09
#define SABM_FRAME 0x2f
#define DM_FRAME 0x0f
#define DISC_FRAME 0x43
#define UA_FRAME 0x63
#define FRMR_FRAME 0x87

/* Control Field Values. */
#define SUPER_FRAME 0x01
#define UNNUM 0x03
#define LOWER_HALF 0x0f
#define FC_MODULO 8 /* Flow Control Modulo value for N(r) and N(s) increments. */

/* FRMR Frame Values (Extra Fields). */
#define DEF_REJD_CNTL 0xff
#define DEF_WXYZ 0x00

/* Control Field P/F Bit Values. */
#define PF_CLEAR 0x00
#define PF_SET 0x10
```

Program Listing - PROCESS_H.Include

```
/* OPTIONAL Information Mask. */
/* #define INFO_MASK 0x01 */

#define COMMAND      0x00
#define RESPONSE     0x01
#define UNKNOWN      0xfc
#define OTHER_MENU   0xfd
#define QUIT          0xfe
#define EXIT          0xff

/* PF Bit Switch Values. */
#define P_BIT 0x00
#define F_BIT 0x01

/* Array Sizes. */
#define PACKET_MAX 128      /* MAXIMUM packet size. */
#define USER_WIDTH 79      /* MAXIMUM Packet screen user data field width. */
#define ARGLIST_SIZE 5      /* Spawn arg list array size. */

/*****
*                               Spawned Processes Priorities                               *
*                               *                                                         *
*      1.  PROCESS3 higher than PROCESS2.                                           *
*      2.  PROCESS2 higher than PROCESS1.                                           *
*****/
#define PSOS2_PRIO 150
#define PSOS3_PRIO 155

/* Miscellaneous Constants. */
#ifndef DEFAULT
#define DEFAULT 0xff      /* sendframe parameter. */
#endif

#define STOPCHAR 'x'

#define VIEW_DELAY (unsigned long)50      /* Delay for viewing error messages. */
```

Collecting Statistics

This program counts three types of BOP frame check sequences (FCS) - good, abort, and bad. The data source can be either the line or a data file. For each FCS type it displays the following information.

- The total number the FCS type counted.
- The percentage of all the FCS types processed.
- A simple bar chart.
- Three types of time stamps.

This program can serve as an example for creating your own program for collecting other statistics or you can modify this file to meet your requirements. *Do not* modify the Solution disc *CUSTOM_COU.csource* file. This is the *file master copy* and should only be used to install the file on the hard disc.

The *CUSTOM_COU.csource* file is the only file required to run this program.

Running the Program

Running this program involves the following steps.

1. Configure the analyzer using the Setup, Simulate, and Run menus to match the data source, either line or buffer (i.e., file).
2. Create the *CUSTOM_COU.program* using the DataCommC *Make* function.
3. Run the program using DataCommC *C Manager* functions (see *DataCommC User's Guide*).
4. Define the data source as described in "Selecting the Data Source," below.

Statistics Program Display

The program displays the statistics in the same format (see figure 5-1) regardless of the data source. There are three statistics displayed for each FCS type.

- The total number of the FCS type processed.
- The percentage of FCS type processed relative to the *all the FCS processed*. This percentage is accurate to $\pm 0.5\%$.
- A bar graph with each unit representing 25 FCS processed. When there is not enough screen space to continue expanding a bar graph the message "Bar char no longer valid - screen too small - percentages still valid." is displayed.

The three time stamps provide the following information.

Measurement start time	The real time stamp of the first FCS frame processed.
Last frame time stamp	The real time stamp of the last FCS frame processed.
Last bad FCS occurrence	The real time stamp of the last bad FCS frame processed.

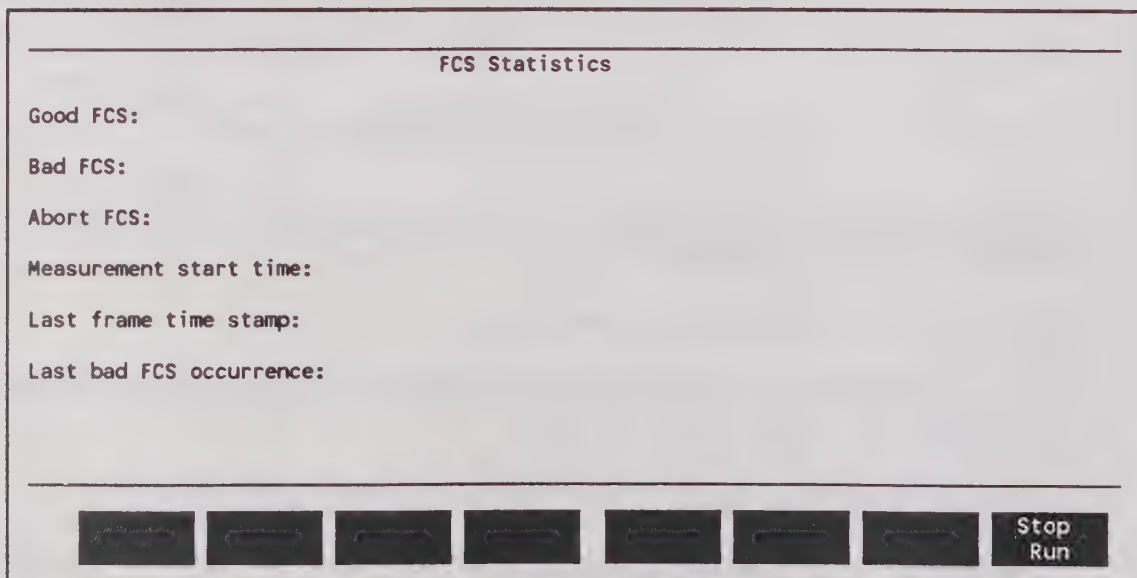


Figure 5-1. FCS Statistics Display

Selecting the Data Source

When the program starts the prompt "Please strike a softkey for the data source." and three softkeys are displayed.

**On
Line**

Selects the line as the data source.

**Buffer
Data**

Selects a data file as the data source.

Exit

Exit the program.

On Line Data Source

When you press **On Line** the program begins counting FCS frames and displays the statistics. If there are no error conditions, the program continues to collect data until you press **Stop Data**. The display is "frozen" for your analysis.

Error conditions which can prevent starting the data collection or cause it to abort include the following.

- Could not communicate with front end hardware.
- Front end is already running.
- Problem setting data source to be on line.
- Start_data() returned something unexpected.
- There is no pod attached.
- Unrecognized front end data source.

Buffer Data Source

When you press **Buffer Data** the **Stop Run** softkey is displayed as is the following prompt.

Please type in the device, path, filename, and file type.
c:\C/Demo/HDLC_data.menu&data

The second line is the default value used by the Demonstration Program. To analyze this file press **Return**.

To display data from another file, press any other key and then enter the complete path name and press **Return**.

The statistics display is the same as for line data. However, the statistics display stops whenever you press **Stop Run** or the end of the data file is reached. In the second case, the message "The run is complete" is displayed.

Error conditions that can affect using buffer data include the following.

- Error opening the data file.
- Problem setting data source to be buffer data.
- Read_message() returned something unexpected.
- Release_event() could not find event.
- Start_data() returned something unexpected.

Source File Listing

The following *CUSTOM_COU.csource* source listing has been reformatted for easier reading and the comments expanded to better describe program functions and operations. *The program code in these listings and the program code on the disc are identical at the time of publication.* The comments and organization are not identical.

As with all software products, the programs may be revised prior to shipment. Therefore, it is possible that there may be *minor* differences.

Program Listing - CUSTOM_COU.csource

```
/******  
*                                     HP 18320A DataCommC                       *  
*                                     Statistics                               *  
*                                     *  
*                                     File: CUSTOM_COU.csource                 *  
*                                     *  
*                                     Copyright 1989                         *  
*                                     Hewlett-Packard Company                 *  
*                                     *  
* This program collect statistics by reading BOP frames and counting the FCS types *  
* (i.e., good, bad, abort). The data source can be either the line or a disc data *  
* file. *  
* Before starting the data collection, configure the HP 4954A Protocol Analyzer *  
* Setup, Simulate, and Run menus for the data source. *  
*****/  
/
```

Program Listing - CUSTOM_COU.csource (Continued)

```

#include <video.h>          /* The video attributes. */
#include <stdio.h>          /* The standard i/o. */
#include <conio.h>          /* The softkey definitions. */
#include <retval.h>        /* The return values. */
#include <dlib.h>           /* DataCommC definitions. */
#include <message.h>        /* All the messaging things. */
#include <setjmp.h>         /* Used for setjmp and longjmp. */

#define INPUT_STRING        0x32  /* Input string length. */
#define FIRST_ROW          0x01  /* The first row of the screen. */
#define FIRST_COLUMN       0x01  /* The first screen column. */
#define LAST_ROW           0x19  /* The last row of the screen. */
#define LONGJMP_VALUE      0x01  /* The value parameter for longjmp. */
#define WAIT_TIME          1L    /* How long to wait for a message. */
#define PERCENT            100   /* How to create a percent value. */
#define X_NUMBER           0x19  /* How often to display bar graph data. */
#ifdef BACKSPACE
#undef BACKSPACE
#endif

#define BACKSPACE           0x15  /* The backspace character. */

#ifdef LINEFEED
#undef LINEFEED
#endif

#define LINEFEED            0x0a  /* The linefeed character. */

#ifdef WAIT
#undef WAIT
#endif
/* Cancels the NO_WAIT macro definition. */

#define WAIT                1     /* NO_WAIT is zero for read message. */

void put_all_sks();
void center();
void set_attribute();
void wipe();
void set_cursor();
void putch();
void decode();
void get_cursor();
void cursor_on();
void cursor_off();
jmp_buf envbuf;           /* Used by setjmp to save registers. */

```

Program Listing - CUSTOM_COU.csource (Continued)

```

/*****
* The "main" routine determines the user-defined data source and checks for error
* conditions.
*****/

main()
{
    int soft_key_input;          /* Which softkey has been input. */
    int check;                  /* Used to check return values. */
    char filename[INPUT_STRING + 1]; /* Used to hold the filename. */
    int counter;                /* A loop counter. */
    int row, column;            /* The rows and columns of a get_cursor. */

    /*
    setjmp(envbuf);              /* Store the registers. */

    while (TRUE)                /* Start while. */
    {
        cursor_off();
        wipe(FIRST_ROW, (LAST_ROW - 0x3));
        set_attribute(NORMAL);
        center("Please strike a softkey for the data source", FIRST_ROW);
        put_all_sks("On 'Line'", "Buffer 'Data'", "", "", "", "", "", "", "Exit", INVERSE);
        soft_key_input = getch();
        wipe((LAST_ROW - 0x2), (LAST_ROW - 0x2)); /* Clear the last line. */

        switch (soft_key_input) /* Start of Softkey switch. */
        {
            case SK1 : check = set_data_source(LINE_RUN, "Bogus string");
                if (check != SUCCESSFUL)
                {
                    center("Problem setting data source to be on line", (LAST_ROW - 0x2));
                    longjmp(envbuf, LONGJMP_VALUE);
                }
                decode();
                break;

            case SK2 : put_all_sks("", "", "", "", "", "", "", "", "", INVERSE);
                set_cursor((LAST_ROW - 0x0a), FIRST_COLUMN);
                cursor_on();
                printf("Please type in the device, path, filename, and file type:\n");
                strcpy(filename, "c:C/Demo/HDLC_data.menus&data");
                printf("%s", filename);
                if ((getch()) != LINEFEED)
                {
                    set_cursor((LAST_ROW - 0x09), FIRST_COLUMN);
                    wipe((LAST_ROW - 0x09));
                    for (counter = NULL; counter < (INPUT_STRING + 1); counter++)
                        filename[counter] = NULL;
                    for (counter = NULL; counter < INPUT_STRING; counter++)
                    {
                        filename[counter] = (char) getch();
                        while (filename[counter] == BACKSPACE)

```

Program Listing - CUSTOM_COU.csource (Continued)

```

        {
            if (counter == NULL)
            {
                filename[NULL] = NULL;
                counter--;
            }
            else
            {
                get_cursor(&row,&column);
                set_cursor(row,(column - 1));
                putchar(' ');
                set_cursor(row,(column - 1));
                filename[counter] = NULL;
                counter--;
                filename[counter] = (char) getch();
            }
        }
        if(filename[counter] == LINEFEED)
        {
            filename[counter] = NULL;
            break;
        }
        if (counter >= NULL)
            putchar(filename[counter]);
    }
    check = set_data_source(DISC_RUN,filename);
    if (check != SUCCESSFUL)
    {
        center("Problem setting data source to be buffer data",(LAST_ROW - 0x2));
        longjmp(envbuf, LONGJMP_VALUE);
    }
    decode();
    break;

case SK8 : goto QUIT;
        break;

default : break;
    }
}
QUIT:
;
}
/* End of Softkey switch. */
/* End of while. */

/* End of main. */

```

Program Listing - CUSTOM_COU.csource (Continued)

```

/*****
* The "decode" routine determines the FCS type, the number of frames, the
* percentage, the time stamp for each event, and displays this information.
*****/
void decode() /* start decode routine. */

{
    int key_input; /* Hold softkey inputs. */
    MESSAGE message; /* Storage for a message. */
    FRAME_EVENT *frame_event; /* Used to easily access a frame event. */
    long good_fcs = 0; /* Holds the good FCS. */
    long abort_fcs = 0; /* Holds the abort FCS. */
    long bad_fcs = 0; /* Holds the bad FCS. */
    int check; /* Used to check return values. */
    long total_frames = 0; /* Stores the total number of frames. */
    int start_flag = 0; /* Used to flag the first frame. */
    int good_column = 0x1d; /* Good FCS bar graph Xs. */
    int bad_column = 0x1d; /* Bad FCS bar graph Xs. */
    int abort_column = 0x1d; /* Abort FCS bar graph Xs. */

    cursor_off();
    wipe(FIRST_ROW, LAST_ROW);
    put_all_sks("", "", "", "", "", "", "", "", "Stop 'Run", INVERSE);
    center("-----");
    center("----", FIRST_ROW);
    center("-----");
    center("----", (LAST_ROW - 0x3));
    center("FCS Statistics", (FIRST_ROW + 0x1));
    set_cursor((FIRST_ROW + 0x3), FIRST_COLUMN);
    printf("Good FCS:\n");
    printf("\n");
    printf("Bad FCS:\n");
    printf("\n");
    printf("Abort FCS:\n");
    printf("\n");
    printf("Measurement start time:\n");
    printf("\n");
    printf("Last frame time stamp:\n");
    printf("\n");
    printf("Last bad FCS occurrence:\n");

    check = start_data();
    if (check != SUCCESSFUL)
    {
        switch (check) /* Start of error switch. */
        {
            case ERROR_1 : center("Error opening the data file", (LAST_ROW - 0x2));
                           goto QUITWHILE1;

            case ERROR_2 : center("Unrecognized front end data source", (LAST_ROW - 0x2));
                           goto QUITWHILE1;

            case ERROR_3 : center("Front end is already running", (LAST_ROW - 0x2));
                           goto QUITWHILE1;
        }
    }
}

```

Program Listing - CUSTOM_COU.csource (Continued)

```

        case ERROR_9 : center("There is no pod attached",(LAST_ROW - 0x2));
                        goto QUITWHILE1;

        case ERROR_10 : center("Could not communicate with front end
                                hardware",(LAST_ROW - 0x2));
                        goto QUITWHILE1;

        default : center("Start_data() returned something unexpected",
                           (LAST_ROW - 0x2));
                  goto QUITWHILE1;
    }

while(TRUE)
{
    if (is_key_avail())
    {
        key_input = getch();
        if (key_input == SK8)
            goto QUITWHILE;
    }

    check = read_message(&message, WAIT, WAIT_TIME);
    switch (check)
    {
        case SUCCESSFUL : if (message.subtype == END_OF_DISC_RUN)
                            goto QUITWHILE;

        else
            if ((message.subtype == DTE_FRAME) || (message.subtype == DCE_FRAME))
            {
                frame_event = message.body.event.event_ptr;
                if (!start_flag)
                {
                    start_flag++;
                    set_cursor((FIRST_ROW + 0x9), (FIRST_COLUMN + 0x1a));
                    printf("%02d:%02d:%03u", frame_event->hour, frame_event->minute,
                           frame_event->second, frame_event->msec);
                }
                set_cursor((FIRST_ROW + 0xb), (FIRST_COLUMN + 0x1a));
                printf("%02d:%02d:%03u", frame_event->hour, frame_event->minute,
                       frame_event->second, frame_event->msec);
                total_frames++;
                if (frame_event->fcs == GOOD_FCS)
                {
                    good_fcs++;
                    if (((good_fcs % X_NUMBER) == NULL) && (good_fcs != NULL))
                    {
                        if (good_column < 0x33)
                        {
                            set_cursor((FIRST_ROW + 0x3), good_column++);
                            set_attribute(SPECIAL);
                            printf("%c", 0xf1);
                        }
                    }
                }
            }
    }
}

```

Program Listing - CUSTOM_COU.csource (Continued)

```

        else
            if (good_column == 0x33)
                center("Bar chart no longer valid - screen too small
                    - percentages still valid",23);
        }
        set_cursor((FIRST_ROW + 0x3),(FIRST_COLUMN + 0xd));
        printf("%ld",good_fcs);
        set_cursor((FIRST_ROW + 0x3),(FIRST_COLUMN + 0x3f));
        printf("%-5.1f%%",(((double) good_fcs) / (double) total_frames) *
            (double) PERCENT));
    }
    else
        if (frame_event->fcs == BAD_FCS)
        {
            bad_fcs++;
            if (((bad_fcs % X_NUMBER) == NULL) && (bad_fcs != NULL))
            {
                if (bad_column < 0x33)
                {
                    set_cursor((FIRST_ROW + 0x5),bad_column++);
                    set_attribute(SPECIAL);
                    printf("%c",0xf1);
                }
                else
                    if (bad_column == 0x33)
                        center("Bar chart no longer valid - screen too small -
                            percentages still valid",23);
            }
            set_cursor((FIRST_ROW + 0x5),(FIRST_COLUMN + 0xd));
            printf("%ld",bad_fcs);
            set_cursor((FIRST_ROW + 0xd),(FIRST_COLUMN + 0x1a));
            printf("%02d:%02d:%02d:%03u",frame_event->hour,
                frame_event->minute,frame_event->second,frame_event->msec);
            set_cursor((FIRST_ROW + 0x5),(FIRST_COLUMN + 0x3f));
            printf("%-5.1f%%",(((double) bad_fcs) / (double) total_frames) *
                (double) PERCENT));
        }
    else
        if (frame_event->fcs == ABORT_FCS)
        {
            abort_fcs++;
            if (((abort_fcs % X_NUMBER) == NULL) && (abort_fcs != NULL))
            {
                if (abort_column < 0x33)
                {
                    set_cursor((FIRST_ROW + 0x7),abort_column++);
                    set_attribute(SPECIAL);
                    printf("%c",0xf1);
                }
                else
                    if (abort_column == 0x33)
                        center("Bar chart no longer valid - screen too small
                            - percentages still valid",23);
            }
        }
    }

```

Program Listing - CUSTOM_COU.csource (Continued)

```

        set_cursor((FIRST_ROW + 0x7),(FIRST_COLUMN + 0xd));
        printf("%ld",abort_fcs);
        set_cursor((FIRST_ROW + 0x7),(FIRST_COLUMN + 0x3f));
        printf("%-5.1f%%",(((double) abort_fcs) / (double)
                                total_frames) * (double) PERCENT));
    }
    if ((message.subtype == DTE_FRAME) || (message.subtype == DCE_FRAME)
        || (message.subtype == LEAD_CHANGE))
    {
        check = release_event(message.body.event.event_ptr);
        if (check != SUCCESSFUL)
        {
            center("Release_event() could not find event",(LAST_ROW - 0x2));
            goto QUITWHILE1;
        }
    }
    break;
    case ERROR_1 : break;
    default : center("Read_message() returned something unexpected",
                    (LAST_ROW - 0x2));
              goto QUITWHILE1;
    }
}

QUITWHILE:
    wipe((LAST_ROW - 0x2),(LAST_ROW - 0x2));
    center("The run is complete",(LAST_ROW - 0x2));

QUITWHILE1:

    stop_data();

    wipe((LAST_ROW - 0x1),LAST_ROW);
    put_all_sks("", "", "", "", "", "", "", "", "Exit 'Menu", INVERSE);

    while(TRUE)
        if (is_key_avail())
        {
            wipe((LAST_ROW - 0x2),(LAST_ROW - 0x2));
            key_input = getch();
            if (key_input == SK8)
                return();
        }
    }

/* End of decode routine. */

```


HP
BAGLEY

THE
BAGLEY



Copyright © 1990
Hewlett-Packard
Printed in USA 9/90

Reorder No. or
Manual Part No.
18320-99506



18320-99506E0990

31 21 10
PA 10 10





9282-1148

Binder Only Part No.

Made in U.S.A.